Contents

Chapter 1

Source Code

This is the full source code of the 68k/Sim project, properly typeset for increased readability. Part I describes the backend (CPU class, memory class, S-record parsing, instruction decoding, execution and translation, etc.) and part II describes the frontend (the Qt code for the graphical user interface).

Part I

Backend

1.1 68sim.cpp

This is the main application file for the console mode of the program (known to insiders as the "Phase 1 mode"). It will only be compiled if a PHASE1 macro is defined (compile option -DPHASE1).

```
/*
   Motorola 68008 Simulator
   (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
   Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

   This is the main application module for phase 1
   This will only be compiled when PHASE1 is defined
*/

#ifdef PHASE1

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "M68008.h"

M68008 *m68008;
MemDevice *memory;

Stack<char*> helpStack;
Stack<unsigned long int> pushPop;
Stack<stackItem> systemStack;

int main( int argc, char **argv, char **env )
{
   FILE *in;
   unsigned long pc;
   char *t;

   char buf[ 32 ];
   unsigned long address;

   // Welcome
      "(C) Trinity College Dublin, 2002\n\n" );

   // Check for number of parameters
   if( argc != 4 )
   {
      printf( "Usage:   68sim filename initial_pc [exec|decode]\n"
         "Example: 68sim test.h68 0x400 exec\n" );
      return 0;
   }

   // Open the input file
   in = fopen( argv[ 1 ], "rw" );
   if( !in )
   {
```

```
        printf( "Error: could not open file '%s' for input.\n", argv [ 1 ] ) ;
        return -1;
    }

    // Initialize memory (1 MB or 0x100000 bytes)
    memory = new MemDevice( 1048576 );

    // Load into memory
    if ( !memory->LoadSRecords( in ))
    {
        printf( "Error: '%s' is not in valid S-Record format.\n", argv[ 1 ] );
        return -1;
    }

    // Close the input file again
    fclose ( in ) ;

    // Convert the second parameter (initial pc) to an integer
    pc = strtol ( argv [ 2 ], &t , 0 ) ;
    if( *t )
    {
        printf( "Error: Invalid initial PC\n" );
        return -1;
    }

    // Initialize the CPU
    m68008 = new M68008( memory, pc, 0x100000 );

    if ( !strcmp( argv [ 3 ], " exec" ) )
    {
        // And step through the program until the CPU is halted
        // (currently when the instruction $F000 is encountered)
        while( !m68008->IsHalted() )
        {
            m68008->Step();
        }
    }
    else if ( !strcmp( argv [ 3 ], " decode" ) )
    {
        // Decode the program
        address = pc;
        while( !m68008->IsHalted() )
        {
            memset( buf, 0, 32 );
            printf( "$%lX: ", address );
            address = m68008->Translate( address, buf );
            printf( "%s\n", buf );
        }
    } else
        printf( "ERROR: Invalid mode. Choose exec or decode\n" );

    // Shutdown
```

```
    return 0;
}

#endif
```

1.2 M68008.h

The header file describing the CPU class. Over time, this has become the main header file of the project, and it defines a number of additional classes and externs as well.

```
/*
    Motorola 68008 Simulator
    (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
    Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

    This class represents the Motorola 68008 Microprocessor
*/

#ifndef M68008_H
#define M68008_H

#include "MemDevice.h"
#include "stack.cpp"
#include "stackItem.h"

#ifdef M68K_LITTLE_ENDIAN
#define DATA_REGISTER( reg )union { struct{ unsigned char b1, b2, b3; unsigned char b; }; struct{
    unsigned short int w1; unsigned short int w; }; unsigned long l; } reg
#define ADDRESS_REGISTER( reg )union { struct{ unsigned short int w1; unsigned short int w; };
    unsigned long l; } reg
#else
#define DATA_REGISTER( reg )union { unsigned char b; unsigned short int w; unsigned long l; } reg
#define ADDRESS_REGISTER( reg )union { unsigned short int w; unsigned long l; } reg
#endif

#define PUSH(origin,size) {int i; if( pushStack.getCount() )pushStack.setTop( true ); mState->stackOp =
    size; for(i=size-1;i>=0;i--) systemStack.push(new StackItem(maxID, m68008->a[7].l+i, origin)); maxID
     += 1;}
#define POP( size ){int i; for(i=0;i<size;i++) mState->push( systemStack.pop() ); if(systemStack.getCount
    ()==0){maxID=0;}else{maxID = ((systemStack.getTop())->ID)+1;} }

#define ILL_INSTR 0xFFFFFFFF

struct CCR
{
    unsigned char HB;
    unsigned char reserved : 3;
    unsigned char x : 1;
    unsigned char n : 1;
    unsigned char z : 1;
    unsigned char v : 1;
```

```
    unsigned char c : 1;
};

struct CCC
{
    // Structure for Condition Code Computations
    unsigned char Sm : 1; // MSB Source Operand
    unsigned char Dm : 1; // MSB Destination Operand
    unsigned char Rm : 1; // MSB Result
};

class MState {
public:
    // Registers
    DATA_REGISTER( d[ 8 ] );   // Data registers
    ADDRESS_REGISTER( a[ 8 ] ); // Address registers
    CCR ccr;                   // Condition Code Register
    unsigned long pc;          // Program counter
    unsigned short int ir;     // Instruction register
    bool halted;               // Halted?

    // Memory
    unsigned long memData[ 20 ]; // At most 16 changes in memory per instruction (movem), 20 for safety's
        sake ;-)
    unsigned long memAddr[ 20 ];
    int numChanges;

    // Stack operation
    int stackOp;               // How many push instructions
    StackItem *stackItem[8];   // We can undo up to 8 (quadword) pops
    int stackItemNr;           // Size of the pop (if any)

    void push( StackItem *sI )
    {
        stackItem[ stackItemNr ] = sI;
        stackItemNr++;
    }
};

class M68008
{
private:
    MemDevice *memory, *bMemory;
    bool halted;
    unsigned long memsize;

    MState* CopyState();
    void SaveState( MState* );
    void Link();


public:
```

```
    M68008( MemDevice *_memory, MemDevice *_bMemory, unsigned long _memsize );

    Stack<MState*> undoStack;
    void Step();
    unsigned long Translate( unsigned long, char * );
    void Halt();
    void PrintRegisters();
    void PrintStr();
    void ReadStr();
    void Reset();
    void SetInitPC( unsigned long int );
    bool RestoreState();

    bool IsHalted();

public:
    /* Register Set */
    DATA_REGISTER( d[ 8 ] );   // Data registers
    ADDRESS_REGISTER( a[ 8 ] ); // Address registers
    CCR ccr;                   // Condition Code Register
    unsigned long pc;          // Program counter
    unsigned short int ir;     // Instruction register
    unsigned long initPC;      // Initial program counter
    const char* currentInstruction;     // String representation of the current instruction
    bool decImm;

    /* Instruction set (in alphabetical order, privileged instructions omitted) */
    void abcd();       // Tested
    void add();        // Tested
    void addi();       // Tested
    void addq();       // Tested
    void addx();       // Tested
    void and();        // Tested
    void andi();       // Tested
    void andiCcr();      // Tested
    void asl_rReg();   // Tested
    void asl_rMem();   // Tested
    void bxx();        // Tested
    void bchgStat();
    void bchgDyn();
    void bclr(unsigned char Bit);
    void bclrStat();
    void bclrDyn();
    void bkpt();         // Tested
    void bra();        // Tested
    void bset(unsigned char Bit);
    void bsetStat();
    void bsetDyn();
    void bsr();        // Tested
    void btst(unsigned char Bit);
    void btstStat();
    void btstDyn();
```

```
void chk();          // Tested
void clr();          // Tested
void cmp();          // Tested
void cmpa();         // Tested
void cmpi();         // Tested
void cmpm();         // Tested

void dbxx();         // *
void divsWord();     //
void divsLong();     // invalid size
void divuWord();     //
void divuLong();     // invalid size
void eor();          // *
void eori();         // *
void eoriCcr();      // *
void exg();          // *
void ext();          // *
void illegal();      // *
void jmp();          // *
void jsr();          // *
void lea();          // *
void linkWord();     // *
void linkLong();     // invalid size
void lsl_rReg();     // *
void lsl_rMem();     // *
void move();         // *
void movea();        // *
void moveFromCcr();  // *
void moveToCcr();    // *
void movem();        //
void movep();
void moveq();        // *
void mulsWord();     // * unsure about the n flag
void mulsLong();     //  invalid size
void muluWord();     // *
void muluLong();     //  invalid size
void nbcd();         //
void neg();          // *
void negx();         // *
void nop();          // *
void not();          // *

void or();           // *
void ori();          // *
void oriCcr();       // *
void pea();          // *
void rol_rReg();     // *
void rol_rMem();
void roxl_rReg();    // *
void roxl_rMem();
void rts();          // *
void sbcd();
```

```
void sxx();
void sub(); // *
void suba(); // *
void subi(); // *
void subq(); // *
void subx(); // *
void swap(); // *
void tas();
void trap(); // *
void trapv(); // *
void tst();
void unlk(); // *

// The methods to translate the instruction into a string
unsigned long tS_abcd( unsigned long, char * );
unsigned long tS_add( unsigned long, char * );
unsigned long tS_addi( unsigned long, char * );
unsigned long tS_addq( unsigned long, char * );
unsigned long tS_addx( unsigned long, char * );
unsigned long tS_and( unsigned long, char * );
unsigned long tS_andi( unsigned long, char * );
unsigned long tS_andiCcr( unsigned long, char * );
unsigned long tS_asl_rReg( unsigned long, char * );
unsigned long tS_asl_rMem( unsigned long, char * );
unsigned long tS_bxx( unsigned long, char * );
unsigned long tS_bchgStat( unsigned long, char * );
unsigned long tS_bchgDyn( unsigned long, char * );
unsigned long tS_bclrStat( unsigned long, char * );
unsigned long tS_bclrDyn( unsigned long, char * );
unsigned long tS_bkpt( unsigned long, char * );
unsigned long tS_bra( unsigned long, char * );
unsigned long tS_bsetStat( unsigned long, char * );
unsigned long tS_bsetDyn( unsigned long, char * );
unsigned long tS_bsr( unsigned long, char * );
unsigned long tS_btstStat( unsigned long, char * );
unsigned long tS_btstDyn( unsigned long, char * );
unsigned long tS_chk( unsigned long, char * );
unsigned long tS_clr( unsigned long, char * );
unsigned long tS_cmp( unsigned long, char * );
unsigned long tS_cmpa( unsigned long, char * );
unsigned long tS_cmpi( unsigned long, char * );
unsigned long tS_cmpm( unsigned long, char * );

unsigned long tS_bits( unsigned long _address, char *buf, char* Instr, const char* Arg1 );

unsigned long tS_dbxx( unsigned long, char * );
unsigned long tS_divsWord( unsigned long, char * );
unsigned long tS_divsLong( unsigned long, char * );
unsigned long tS_divuWord( unsigned long, char * );
unsigned long tS_divuLong( unsigned long, char * );
unsigned long tS_eor( unsigned long, char * );
unsigned long tS_eori( unsigned long, char * );
```

```
    unsigned long tS_eoriCcr( unsigned long, char * );
    unsigned long tS_exg( unsigned long, char * );
    unsigned long tS_ext( unsigned long, char * );
    unsigned long tS_illegal( unsigned long, char * );
    unsigned long tS_jmp( unsigned long, char * );
    unsigned long tS_jsr( unsigned long, char * );
    unsigned long tS_lea( unsigned long, char * );
    unsigned long tS_linkWord( unsigned long, char * );
    unsigned long tS_linkLong( unsigned long, char * );
    unsigned long tS_lsl_rReg( unsigned long, char * );
    unsigned long tS_lsl_rMem( unsigned long, char * );
    unsigned long tS_move( unsigned long, char * );
    unsigned long tS_movea( unsigned long, char * );
    unsigned long tS_moveFromCcr( unsigned long, char * );
    unsigned long tS_moveToCcr( unsigned long, char * );
    unsigned long tS_movem( unsigned long, char * );
    unsigned long tS_movep( unsigned long, char * );
    unsigned long tS_moveq( unsigned long, char * );
    unsigned long tS_mulsWord( unsigned long, char * );
    unsigned long tS_mulsLong( unsigned long, char * );
    unsigned long tS_muluWord( unsigned long, char * );
    unsigned long tS_muluLong( unsigned long, char * );
    unsigned long tS_nbcd( unsigned long, char * );
    unsigned long tS_neg( unsigned long, char * );
    unsigned long tS_negx( unsigned long, char * );
    unsigned long tS_nop( unsigned long, char * );
    unsigned long tS_not( unsigned long, char * );

    unsigned long tS_or( unsigned long, char * );
    unsigned long tS_ori( unsigned long, char * );
    unsigned long tS_oriCcr( unsigned long, char * );
    unsigned long tS_pea( unsigned long, char * );
    unsigned long tS_rol_rReg( unsigned long, char * );
    unsigned long tS_rol_rMem( unsigned long, char * );
    unsigned long tS_roxl_rReg( unsigned long, char * );
    unsigned long tS_roxl_rMem( unsigned long, char * );
    unsigned long tS_rts( unsigned long, char * );
    unsigned long tS_sbcd( unsigned long, char * );
    unsigned long tS_sxx( unsigned long, char * );
    unsigned long tS_sub( unsigned long, char * );
    unsigned long tS_suba( unsigned long, char * );
    unsigned long tS_subi( unsigned long, char * );
    unsigned long tS_subq( unsigned long, char * );
    unsigned long tS_subx( unsigned long, char * );
    unsigned long tS_swap( unsigned long, char * );
    unsigned long tS_tas( unsigned long, char * );
    unsigned long tS_trap( unsigned long, char * );
    unsigned long tS_trapv( unsigned long, char * );
    unsigned long tS_tst( unsigned long, char * );
    unsigned long tS_unlk( unsigned long, char * );
};
```

```
extern Stack<char*> helpStack;
extern Stack<unsigned long int> pushPop;
extern Stack<bool> pushStack;
extern M68008 *m68008;
extern MemDevice *memory;
extern Stack<StackItem*> systemStack;
extern MState *mState;
extern int maxID;

#endif
```

1.3 M68008.cpp

The class methods of the CPU class, except for the instruction execution and translation methods which are stored in six seperate files (see below).

```
/*
   Motorola 68008 Simulator
   (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
   Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

   This class represents the Motorola 68008 Microprocessor
*/

#include "M68008.h"
#include <stdio.h>
#include <string.h>
#include <qmessagebox.h>
#include <qinputdialog.h>

#define LINK( mask, opcode, method )if( ( ir & mask )== opcode ){ method(); return; }
#define LINK_EXT( mask1, mask2, opcode1, opcode2, method )if( ( ( ir & mask1 )== opcode1 )&& ( ( op &
    mask2 )== opcode2 )){ method(); return; }

#define TLINK( mask, opcode, method )if( ( ir & mask )== opcode ){ if( address == pc )currentInstruction
    = #method; return method( address, buf ); }
#define TLINK_EXT( mask1, mask2, opcode1, opcode2, method )if( ( ( ir & mask1 )== opcode1 )&& ( ( op
    & mask2 )== opcode2 )){ if( address == pc )currentInstruction = #method; return method( address, buf
    ); }

MState *mState;

// Constructor
M68008::M68008( MemDevice *_memory, MemDevice *_bMemory, unsigned long _memsize )
{
   memsize = _memsize;
   memory = _memory;
   bMemory = _bMemory;
   initPC = 0;

   memory->Clear();
```

```
    Reset();
}

// Reset the CPU and memory
void M68008::Reset()
{
    int lp1;

    halted = false;

    // Clear registers
    for( lp1 = 0; lp1 < 8; lp1++ )
    {
        a[ lp1 ].l = 0;
        d[ lp1 ].l = 0;
    }
    ir = 0;
    ccr.c = 0;
    ccr.n = 0;
    ccr.v = 0;
    ccr.x = 0;
    ccr.z = 0;
    pc = 0;

    // Set SSP to end of memory
    a[ 7 ].l = memsize;

    pc = initPC;
}

// Set the initial and actual PC
void M68008::SetInitPC( unsigned long int _pc )
{
    initPC = _pc;
    pc = _pc;
}

// Execute the next instruction
void M68008::Step()
{
    mState = CopyState();
    Link();
    SaveState( mState );
}

// Output the string in memory pointed to by a[ 0 ] (by opening a messagebox)
void M68008::PrintStr()
{
    QString str;

    for( int lp1 = a[ 0 ].l; memory->GetByte( lp1 )!= 0; lp1++ )
    {
```

```
        str.append( memory->GetByte( lp1 ));
    }

    QMessageBox::information( NULL, "Output from program", str );

    pc += 2;
}

// Read a string, and store the result in memory at position a[ 0 ]
// Prints out a custom message from memory pointed to by a[ 1 ]
void M68008::ReadStr()
{
    QString str, str2;
    QChar c;
    unsigned int lp1;

    for( lp1 = a[ 1 ].l; memory->GetByte( lp1 )!= 0; lp1++ )
    {
        str2.append( memory->GetByte( lp1 ));
    }

    str = QInputDialog::getText( "Motorola 68008 Simulator", str2 );

    for( lp1 = 0; lp1 < str.length(); lp1++ )
    {
        c =  str.latin1()[ lp1 ];
        memory->SetByte( c, a[ 0 ].l + lp1 );
    }
    memory->SetByte( 0, a[ 0 ].l + lp1 );

    pc += 2;
}

// Perform the actual instruction execution
void M68008::Link()
{
    unsigned short int op;

    // Fetch the instruction encoding from memory
    ir = memory->GetWord( pc );
    op = memory->GetWord( pc + 2 );

    // The following instructions are specific to this (virtual) processor
    LINK( 0xFFFF, 0xF000, Halt );
    LINK( 0xFFFF, 0xF001, PrintRegisters );
    LINK( 0xFFFF, 0xF002, PrintStr );
    LINK( 0xFFFF, 0xF003, ReadStr );

    // The following instructions require the IR and the operands to
    // determine which instruction to execute
    LINK_EXT( 0xFFFF, 0xFF00, 0x0A3C, 0x0000, eoriCcr );
    LINK_EXT( 0xFFFF, 0xFF00, 0x023C, 0x0000, andiCcr );
```

```
LINK_EXT( 0xFFFF, 0xFF00, 0x003C, 0x0000, oriCcr );
LINK_EXT( 0xFFC0, 0xFF00, 0x08C0, 0x0000, bsetStat );
LINK_EXT( 0xFFC0, 0xFF00, 0x0880, 0x0000, bclrStat );
LINK_EXT( 0xFFC0, 0xFF00, 0x0840, 0x0000, bchgStat );
LINK_EXT( 0xFFC0, 0xFF00, 0x0800, 0x0000, btstStat );
LINK_EXT( 0xFFC0, 0x8BF8, 0x4C40, 0x0800, divsLong );
LINK_EXT( 0xFFC0, 0x8BF8, 0x4C40, 0x0000, divuLong );  // Note: Incorrect value in CPUREF.PDF
LINK_EXT( 0xFFC0, 0x8BF8, 0x4C00, 0x0800, mulsLong );
LINK_EXT( 0xFFC0, 0x8BF8, 0x4C00, 0x0000, muluLong );

// The following instructions only need the IR
LINK( 0xFFFF, 0x4E76, trapv );
LINK( 0xFFFF, 0x4E75, rts );
LINK( 0xFFFF, 0x4E71, nop );
LINK( 0xFFFF, 0x4AFC, illegal );
LINK( 0xFFF8, 0x4E58, unlk );      // Note: Incorrect value in CPUREF.PDF
LINK( 0xFFF8, 0x4E50, linkWord );
LINK( 0xFFF8, 0x4848, bkpt );
LINK( 0xFFF8, 0x4840, swap );
LINK( 0xFFF8, 0x4808, linkLong );
LINK( 0xFFF0, 0x4E40, trap );
LINK( 0xFFC0, 0x4EC0, jmp );
LINK( 0xFFC0, 0x4E80, jsr );
LINK( 0xFFC0, 0x4AC0, tas );
LINK( 0xFFC0, 0x4840, pea );
LINK( 0xFFC0, 0x4800, nbcd );
LINK( 0xFFC0, 0x44C0, moveToCcr );
LINK( 0xFFC0, 0x42C0, moveFromCcr );
LINK( 0xFF00, 0x6100, bsr );
LINK( 0xFF00, 0x6000, bra );
LINK( 0xFF00, 0x4A00, tst );
LINK( 0xFF00, 0x4600, not );
LINK( 0xFF00, 0x4400, neg );
LINK( 0xFF00, 0x4200, clr );
LINK( 0xFF00, 0x4000, negx );
LINK( 0xFF00, 0x0C00, cmpi );
LINK( 0xFF00, 0x0A00, eori );
LINK( 0xFF00, 0x0600, addi );
LINK( 0xFF00, 0x0400, subi );
LINK( 0xFF00, 0x0200, andi );
LINK( 0xFF00, 0x0000, ori );
LINK( 0xFEC0, 0xE6C0, rol_rMem );
LINK( 0xFEC0, 0xE4C0, roxl_rMem );
LINK( 0xFEC0, 0xE2C0, lsl_rMem );
LINK( 0xFEC0, 0xE0C0, asl_rMem );
LINK( 0xFE38, 0x4800, ext );
LINK( 0xFB80, 0x4880, movem );
LINK( 0xF1F0, 0xC100, abcd );
LINK( 0xF1F0, 0x8100, sbcd );
LINK( 0xF1C0, 0xC1C0, mulsWord );
LINK( 0xF1C0, 0xC0C0, muluWord );
LINK( 0xF1C0, 0x81C0, divsWord );
```

```
    LINK( 0xF1C0, 0x80C0, divuWord );
    LINK( 0xF1C0, 0x41C0, lea );
    LINK( 0xF1C0, 0x01C0, bsetDyn );
    LINK( 0xF038, 0x0008, movep );
    LINK( 0xF1C0, 0x0180, bclrDyn );
    LINK( 0xF1C0, 0x0140, bchgDyn );
    LINK( 0xF1C0, 0x0100, btstDyn );
    LINK( 0xF138, 0xB108, cmpm );
    LINK( 0xF130, 0x9100, subx );
    LINK( 0xF100, 0xC100, exg );
    LINK( 0xF100, 0x7000, moveq );
    LINK( 0xF0F8, 0x50C8, dbxx );
    LINK( 0xF0C0, 0x50C0, sxx );
    LINK( 0xF100, 0x5100, subq );
    LINK( 0xF100, 0x5000, addq );
    LINK( 0xF040, 0x4000, chk );
    LINK( 0xF018, 0xE018, rol_rReg );
    LINK( 0xF018, 0xE010, roxl_rReg );
    LINK( 0xF018, 0xE008, lsl_rReg );
    LINK( 0xF018, 0xE000, asl_rReg );
    LINK( 0xF000, 0xD000, add );    // includes adda
    LINK( 0xF000, 0xC000, and );
    LINK( 0xF130, 0xD100, addx );  // AD: Ensure ADDX is comes after ADD for compatability
    LINK( 0xF000, 0xB000, cmp );   // includes cmpa, also includes eor (included code in cmp to go to eor)
    LINK( 0xF000, 0x9000, sub );    // Note: Incorrect value in CPUREF.PDF, includes suba
    LINK( 0xF000, 0x8000, or );     // Note: Incorrect value in CPUREF.PDF
    LINK( 0xF000, 0x6000, bxx );
    LINK( 0xC1C0, 0x0040, movea );
    LINK( 0xC000, 0x0000, move );

    pc += 2;
}

// CopyState copies the state the current state of the processor and the memory
MState* M68008::CopyState()
{
    MState *mState;
    int lp1;

    mState = new MState;

    // Registers
    mState->halted = halted;
    for( lp1 = 0; lp1 < 8; lp1++ )
    {
        mState->d[ lp1 ].l = d[ lp1 ].l;
        mState->a[ lp1 ].l = a[ lp1 ].l;
    }
    mState->ccr.x = ccr.x;
    mState->ccr.n = ccr.n;
    mState->ccr.z = ccr.z;
    mState->ccr.v = ccr.v;
```

```
    mState->ccr.c = ccr.c;
    mState->pc = pc;
    mState->ir = ir;

    // Memory
    bMemory->Copy( memory );
    mState->numChanges = 0;

    mState->stackOp = 0;
    mState->stackItemNr = 0;

    return mState;
}

// SaveState compares the current state of the processor with the copied state and
// pushes the changes to the Undo stack
void M68008::SaveState( MState *mState )
{
    unsigned long int lp1;

    for( lp1 = 0; lp1 < memsize; lp1 += 4 )
    {
        if( memory->GetLongword( lp1 )!= bMemory->GetLongword( lp1 ))
        {
            mState->memData[ mState->numChanges ] = bMemory->GetLongword( lp1 );
            mState->memAddr[ mState->numChanges ] = lp1;
            (mState->numChanges)++;
        }
    }

    undoStack.push( mState );
}

// RestoreState pops an MState from the undo stack (if any) and restores
// the state of the processor
bool M68008::RestoreState()
{
    MState *mState;
    int lp1;
    StackItem *stackItem;

    if( undoStack.getCount() == 0 )
        return false;

    mState = undoStack.pop();

    // Registers
    halted = mState->halted;
    for( lp1 = 0; lp1 < 8; lp1++ )
    {
        d[ lp1 ].l = mState->d[ lp1 ].l;
        a[ lp1 ].l = mState->a[ lp1 ].l;
```

```
    }
    ccr.x = mState->ccr.x;
    ccr.n = mState->ccr.n;
    ccr.z = mState->ccr.z;
    ccr.v = mState->ccr.v;
    ccr.c = mState->ccr.c;
    pc = mState->pc;
    ir = mState->ir;

    for( lp1 = 0; lp1 < mState->numChanges; lp1++ )
    {
        memory->SetLongword( mState->memData[ lp1 ], mState->memAddr[ lp1 ] );
    }

    if( mState->stackOp )
    {
        for( lp1 = 0; lp1 < mState->stackOp; lp1++ )
        {
            stackItem = systemStack.pop();
            delete stackItem;
        }
        if( systemStack.getCount() )maxID = systemStack.getTop()->ID + 1; else maxID = 0;
    }
    if( mState->stackItemNr )
    {
        for( lp1 = 0; lp1 < mState->stackItemNr; lp1++ )
        {
            systemStack.push( mState->stackItem[ mState->stackItemNr - lp1 - 1 ] );
        }
        if( systemStack.getCount() )maxID = systemStack.getTop()->ID + 1; else maxID = 0;
    }

    delete mState;
    return true;
}

// Translate the instruction at address 'address' to a string.
// The string will be written to buf, which should be 32 bytes long
// Translate will return the address of the next instruction in memory
// (note that this is not necessarily the next instruction to be executed!)
// The return value will be -1 for an unrecognized instruction
unsigned long M68008::Translate( unsigned long address, char *buf )
{
    memset( buf, 0, 32 );

    unsigned short int op, ir;

    // Fetch the instruction encoding from memory
    ir = memory->GetWord( address );
    op = memory->GetWord( address + 2 );

    // Make sure we can still halt the processor (this is not necessary anymore for the GUI-only version)
```

```
if( ir == 0xF000 )
{
    sprintf( buf, "(Internal) Halt processor" );
    #ifdef PHASE1
        m68008->Halt();
    #endif
    return address + 2;
}
// $F001 is used to print the registers to the console; not useful in the GUI mode
if( ir == 0xF001 )
{
    sprintf( buf, "(Internal) Print registers" );
    return address + 2;
}
if( ir == 0xF002 )
{
    sprintf( buf, "(Internal) Print String" );
    return address + 2;
}
if( ir == 0xF003 )
{
    sprintf( buf, "(Internal) Read String" );
    return address + 2;
}

// The following instructions require the IR and the operands to
// determine which instruction to execute
TLINK_EXT( 0xFFFF, 0xFF00, 0x0A3C, 0x0000, tS_eoriCcr );
TLINK_EXT( 0xFFFF, 0xFF00, 0x023C, 0x0000, tS_andiCcr );
TLINK_EXT( 0xFFFF, 0xFF00, 0x003C, 0x0000, tS_oriCcr );
TLINK_EXT( 0xFFC0, 0xFF00, 0x08C0, 0x0000, tS_bsetStat );
TLINK_EXT( 0xFFC0, 0xFF00, 0x0880, 0x0000, tS_bclrStat );
TLINK_EXT( 0xFFC0, 0xFF00, 0x0840, 0x0000, tS_bchgStat );
TLINK_EXT( 0xFFC0, 0xFF00, 0x0800, 0x0000, tS_btstStat );
TLINK_EXT( 0xFFC0, 0x8BF8, 0x4C40, 0x0800, tS_divsLong );
TLINK_EXT( 0xFFC0, 0x8BF8, 0x4C40, 0x0000, tS_divuLong );
TLINK_EXT( 0xFFC0, 0x8BF8, 0x4C00, 0x0800, tS_mulsLong );
TLINK_EXT( 0xFFC0, 0x8BF8, 0x4C00, 0x0000, tS_muluLong );

// The following instructions only need the IR
TLINK( 0xFFFF, 0x4E76, tS_trapv );
TLINK( 0xFFFF, 0x4E75, tS_rts );
TLINK( 0xFFFF, 0x4E71, tS_nop );
TLINK( 0xFFFF, 0x4AFC, tS_illegal );
TLINK( 0xFFF8, 0x4E58, tS_unlk );
TLINK( 0xFFF8, 0x4E50, tS_linkWord );
TLINK( 0xFFF8, 0x4848, tS_bkpt );
TLINK( 0xFFF8, 0x4840, tS_swap );
TLINK( 0xFFF8, 0x4808, tS_linkLong );
TLINK( 0xFFF0, 0x4E40, tS_trap );
TLINK( 0xFFC0, 0x4EC0, tS_jmp );
TLINK( 0xFFC0, 0x4E80, tS_jsr );
```

```
TLINK( 0xFFC0, 0x4AC0, tS_tas );
TLINK( 0xFFC0, 0x4840, tS_pea );
TLINK( 0xFFC0, 0x4800, tS_nbcd );
TLINK( 0xFFC0, 0x44C0, tS_moveToCcr );
TLINK( 0xFFC0, 0x42C0, tS_moveFromCcr );
TLINK( 0xFF00, 0x6100, tS_bsr );
TLINK( 0xFF00, 0x6000, tS_bra );
TLINK( 0xFF00, 0x4A00, tS_tst );
TLINK( 0xFF00, 0x4600, tS_not );
TLINK( 0xFF00, 0x4400, tS_neg );
TLINK( 0xFF00, 0x4200, tS_clr );
TLINK( 0xFF00, 0x4000, tS_negx );
TLINK( 0xFF00, 0x0C00, tS_cmpi );
TLINK( 0xFF00, 0x0A00, tS_eori );
TLINK( 0xFF00, 0x0600, tS_addi );
TLINK( 0xFF00, 0x0400, tS_subi );
TLINK( 0xFF00, 0x0200, tS_andi );
TLINK( 0xFF00, 0x0000, tS_ori );
TLINK( 0xFEC0, 0xE6C0, tS_rol_rMem );
TLINK( 0xFEC0, 0xE4C0, tS_roxl_rMem );
TLINK( 0xFEC0, 0xE2C0, tS_lsl_rMem );
TLINK( 0xFEC0, 0xE0C0, tS_asl_rMem );
TLINK( 0xFE38, 0x4800, tS_ext );
TLINK( 0xFB80, 0x4880, tS_movem );
TLINK( 0xF1F0, 0xC100, tS_abcd );
TLINK( 0xF1F0, 0x8100, tS_sbcd );
TLINK( 0xF1C0, 0xC1C0, tS_mulsWord );
TLINK( 0xF1C0, 0xC0C0, tS_muluWord );
TLINK( 0xF1C0, 0x81C0, tS_divsWord );
TLINK( 0xF1C0, 0x80C0, tS_divuWord );
TLINK( 0xF1C0, 0x41C0, tS_lea );
TLINK( 0xF1C0, 0x01C0, tS_bsetDyn );
TLINK( 0xF038, 0x0008, tS_movep );
TLINK( 0xF1C0, 0x0180, tS_bclrDyn );
TLINK( 0xF1C0, 0x0140, tS_bchgDyn );
TLINK( 0xF1C0, 0x0100, tS_btstDyn );
TLINK( 0xF130, 0x9100, tS_subx );
TLINK( 0xF100, 0xC100, tS_exg );
TLINK( 0xF100, 0x7000, tS_moveq );
TLINK( 0xF0F8, 0x50C8, tS_dbxx );
TLINK( 0xF0C0, 0x50C0, tS_sxx );
TLINK( 0xF100, 0x5100, tS_subq );
TLINK( 0xF100, 0x5000, tS_addq );
TLINK( 0xF040, 0x4000, tS_chk );
TLINK( 0xF018, 0xE018, tS_rol_rReg );
TLINK( 0xF018, 0xE010, tS_roxl_rReg );
TLINK( 0xF018, 0xE008, tS_lsl_rReg );
TLINK( 0xF018, 0xE000, tS_asl_rReg );
TLINK( 0xF000, 0xD000, tS_add );
TLINK( 0xF000, 0xC000, tS_and );
TLINK( 0xF130, 0xD100, tS_addx );
TLINK( 0xF000, 0xB000, tS_cmp );
```

```
    TLINK( 0xF138, 0xB108, tS_cmpm );
    TLINK( 0xF000, 0x9000, tS_sub );
    TLINK( 0xF000, 0x8000, tS_or );
    TLINK( 0xF000, 0x6000, tS_bxx );
    TLINK( 0xC1C0, 0x0040, tS_movea );
    TLINK( 0xC000, 0x0000, tS_move );

    sprintf( buf, "Unknown instruction" );
    //currentInstruction = 0;
    return address + 2;
}

// Halt the processor
void M68008::Halt()
{
    halted = true;
}

// Return TRUE if the processor is halted,
// FALSE otherwise
bool M68008::IsHalted()
{
    return halted;
}

void M68008::PrintRegisters()
{
    int lp1;

    // Print out the PC

    // Print out the CCR

    // Print out data registers
    for( lp1 = 0; lp1 < 4; lp1++ )
        printf( "D%c: %08lX\t", lp1 + '0', d[ lp1 ].l );
    for( lp1 = 4; lp1 < 8; lp1++ )
        printf( "D%c: %08lX\t", lp1 + '0', d[ lp1 ].l );

    // Print out address registers
    for( lp1 = 0; lp1 < 4; lp1++ )
        printf( "A%c: %08lX\t", lp1 + '0', a[ lp1 ].l );
    for( lp1 = 4; lp1 < 8; lp1++ )
        printf( "A%c: %08lX\t", lp1 + '0', a[ lp1 ].l );

    pc += 2;
}
```

1.4 MInstrA-C.cpp

The execution methods of the instructions starting with $A \cdots C$.

```
/*
   Motorola 68008 Simulator
   (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
   Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

   This class represents the Motorola 68008 Microprocessor
   Instruction definitions A-C
*/

#include <stdio.h>
#include "M68008.h"

#define MSB8 0x80
#define MSB16 0x8000
#define MSB32 0x80000000

#define GETBITS(addr, mask, shift) (memory->GetWord(addr) & mask) >> shift
#define GET_INSTRBITS(mask, shift) (ir & mask) >> shift
#define BYTE_MSB(byte) ( byte & 0x80 )>> 7
#define WORD_MSB(word) ( word & 0x8000 )>> 15

#define LONG_MSB(longw) ( longw & 0x80000000 )>> 31
#define SUB_SETV( S, D, R, M )ccr.v = ( ( ~( S & M )& ( D & M )& ~( R & M )| ( S & M )& ~( D & M )
    & ( R & M ))& M )? 1 : 0;
#define SUB_SETC( S, D, R, M )ccr.c = ccr.x = ( ( ( S & M )& ~( D & M )| ( R & M )& ~( D & M )| ( S &
    M )& ( R & M ))& M ) ? 1 : 0;
#define SUB_SETOC( S, D, R, M )ccr.c = ( ( ( S & M )& ~( D & M )| ( R & M )& ~( D & M )| ( S & M )& (
    R & M ))& M )? 1 : 0;
#define SETNZB( src ){ ccr.n = ( (signed char)src < 0 ); ccr.z = ( src == 0 ); }
#define SETNZW( src ){ ccr.n = ( (signed short)src < 0 ); ccr.z = ( src == 0 ); }
#define SETNZL( src ){ ccr.n = ( (signed long)src < 0 ); ccr.z = ( src == 0 ); }

void M68008::abcd()
{
   //
   // Alan Donnelly (16/04/02)
   // ABCD: Rx(b10) + Ry(b10) + ccr.x
   //

   unsigned char RM = GETBITS( pc, 0x8, 3);  // Register or memory ( -(An) )
   unsigned char Rx = GETBITS( pc, 0xE00, 9);  // Destination Register
   unsigned char Ry = GETBITS( pc, 0x7, 0);  // Source Register

   unsigned char LowNybbleSum = 0;          // BCD sum of low nybbles
   unsigned char HighNybbleSum = 0;       // BCD sum of high nybbles

   switch ( RM )
   {
      case 0: // Dy,Dx

         // Sum each nybble
         LowNybbleSum = ( d[ Rx ].b & 0x0F )+ ( d[ Ry ].b & 0x0F )+ ccr.x;
```

```
    HighNybbleSum = ( d[ Rx ].b & 0xF0 )+ ( d[ Ry ].b & 0xF0 );

    // Check for BCD carries
    if ( LowNybbleSum > 9 )
    {
        LowNybbleSum %= 10;
        HighNybbleSum++;
    }

    if ( HighNybbleSum > 9 )
    {
        HighNybbleSum %= 10;
        ccr.c = 1;
    }

    // Finish the BCD sum
    d[ Rx ].b = LowNybbleSum + HighNybbleSum;

    // Check for nonzero result and clear Z if non-zero
    if ( d[ Rx ].b != 0 )
        ccr.z = 0;

    break;

case 1: // -(Ay),-(Ax)
    // Pre-decrement the address registers
    a[ Rx ].l--;
    a[ Ry ].l--;

    // Sum each nybble
    LowNybbleSum = ( memory->GetByte( a[ Rx ].l )& 0x0F )+ ( memory->GetByte( a[ Ry ].l )& 0x0F
        )+ ccr.x;
    HighNybbleSum = ( memory->GetByte( a[ Rx ].l )& 0xF0 )+ ( memory->GetByte( a[ Ry ].l )& 0xF0
        );

    // Check for BCD carries
    if ( LowNybbleSum > 9 )
    {
        LowNybbleSum %= 10;
        HighNybbleSum++;
    }

    if ( HighNybbleSum > 9 )
    {
        HighNybbleSum %= 10;
        ccr.c = 1;
    }

    // Finish the BCD sum
    memory->SetByte( LowNybbleSum + HighNybbleSum, a[ Rx ].l );

    // Check for nonzero result and clear Z if non-zero
```

```
            if ( memory->GetByte( a[ Rx ].l )!= 0 )
                ccr.z = 0;

            break;
        }

        ccr.x = ccr.c;
        pc += 2; // Update PC
}

void M68008::add()
{
        //
        // Alan Donnelly (23/02/2002)
        //

        //
        // This also handles ADDA ( same instruction signature )
        //

        unsigned char Reg = GETBITS(pc, 0xE00, 9);
        unsigned char OpMode = GETBITS(pc, 0x1C0, 6);
        unsigned char EaMode = GETBITS(pc, 0x38, 3);
        unsigned char EaReg = GETBITS(pc, 0x7, 0);
        unsigned long DispedMem = 0;            // Displaced memory address
        DATA_REGISTER( Result );
        DATA_REGISTER( EaValue );
        CCC ccc;

        bool EaIsSource;

        // (d16, An)
        signed short int Disp16;

        // Disp8(An, Xn)
        signed char Disp8;
        unsigned char Xn;
        unsigned char Wl;

        // get addressing mode and source (Ea) value
        EaIsSource = GETBITS(pc, 0x100, 0) == 0;

        switch( OpMode )
        {
        case 0:  // BYTE (Source: Ea)
            ccc.Dm = BYTE_MSB( d[ Reg ].b );

            switch( EaMode )
            {
            case 0:      // Dn.b
                ccc.Sm = BYTE_MSB( d[ EaReg ].b );
```

```
    d[ Reg ].b += d[ EaReg ].b;

    break;
case 1:     // An.b (doesn't apply)
    addx(); return;
    break;
case 2:     // (An).b
case 3:     // (An)+.b
case 4:     // -(An).b
    if( EaMode == 4 )a[ EaReg ].l--;

    EaValue.b = memory->GetByte( a[ EaReg ].l );
    ccc.Sm = BYTE_MSB( EaValue.b );

    d[ Reg ].b += EaValue.b;

    if( EaMode == 3 )a[ EaReg ].l++;
    break;
case 5:     // (d16, An).b
    Disp16 = memory->GetWord( pc + 2 );
    EaValue.b = memory->GetByte( a[ EaReg ].l + Disp16 );
    ccc.Sm = BYTE_MSB( EaValue.b );

    d[ Reg ].b += EaValue.b;

    // modify PC
    pc += 2;

    break;
case 6:     // Disp8(An, Xn).b
    //
    // Immediate data for Address Register Indirect with (8-bit Displacement) Mode
    // +--+--+--+-----+---+---+---+---+--+--+--+--+--+--+--+--+--+
    //|    Xn   | w/l  | 0 | 0 | 0 | 0 |       Disp8         |
    // +--+--+--+-----+---+---+---+---+--+--+--+--+--+--+--+--+--+
    //
    // bits  between Xn and w/l are scale factor (n/a to add instr.)???
    //

    Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
    Xn = GETBITS( pc + 2, 0x7000, 12 );
    Wl = GETBITS( pc + 2, 0x800, 11 );

    if ( !Wl )      // word index
        EaValue.b = memory->GetByte( a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w );
    else
        EaValue.b = memory->GetByte( a[ EaReg ].l + Disp8 + d[ Xn ].l );

    ccc.Sm = BYTE_MSB( EaValue.b );
    d[ Reg ].b += EaValue.b;

    pc += 2;    // modify PC
```

```
    break;
case 7:
    switch( EaReg )
    {
        case 0:         // (www).W
            ccc.Sm = BYTE_MSB( memory->GetByte( memory->GetWord( pc + 2 )));
            d[ Reg ].b += memory->GetByte( memory->GetWord( pc + 2 ));

            pc += 2; // modify PC
            break;
        case 1:         // (www).L
            ccc.Sm = BYTE_MSB( memory->GetByte( memory->GetLongword( pc + 2 )));
            d[ Reg ].b += memory->GetByte( memory->GetLongword( pc + 2 ));

            pc += 4; // modify PC
            break;
        case 2:         // (d16, PC).b
            Disp16 = memory->GetWord( pc + 2 );
            ccc.Sm = BYTE_MSB( memory->GetByte( pc + Disp16 ));
            d[ Reg ].b += memory->GetByte( pc + Disp16 );

            pc += 2; // modify PC
            break;
        case 3:         // d8(PC, Xn)
            Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
            Xn = GETBITS( pc + 2, 0x7000, 12 );
            Wl = GETBITS( pc + 2, 0x800, 11 );

            if ( !Wl )   // word index
                EaValue.b = memory->GetByte( pc + Disp8 + (signed short int)d[ Xn ].w );
            else      // long index
                EaValue.b = memory->GetByte( pc + Disp8 + d[ Xn ].l );

            ccc.Sm = BYTE_MSB( EaValue.b );
            d[ Reg ].b += EaValue.b;

            pc += 2; // modify PC
            break;
        case 4:         // #<data>
            ccc.Sm = BYTE_MSB( memory->GetByte( pc + 3) );
            d[ Reg ].b += memory->GetByte( pc + 3);

            pc += 2; // modify PC
            break;
        case 5:
        case 6:
        case 7:
            addx(); return;
            break;
    }
    break;
```

```
    }

    // Set carry flag variables
    ccc.Rm = BYTE_MSB( d[ Reg ].b );
    ccr.z = d[ Reg ].b == 0;

    break;

case 1:
    ccc.Dm = WORD_MSB( d[ Reg ].w );

    switch( EaMode )
    {
    // WORD (Source: Ea)
    case 0:     // Dn.w
        ccc.Sm = WORD_MSB( d[ EaReg ].w );
        d[ Reg ].w += d[ EaReg ].w;
        break;
    case 1:     // An.w
        ccc.Sm = WORD_MSB( a[ EaReg ].w );
        d[ Reg ].w += a[ EaReg ].w;
        break;
    case 2:     // (An).w
    case 3:     // (An)+.w
    case 4:     // -(An).w
        if( EaMode == 4 )a[ EaReg ].l -= 2;
        EaValue.w = memory->GetWord( a[ EaReg ].l );
        ccc.Sm = WORD_MSB( EaValue.w );

        d[ Reg ].w += EaValue.w;

        if( EaMode == 3 )a[ EaReg ].l += 2;

        break;
    case 5:  // (d16, An).w
        Disp16 = memory->GetWord( pc + 2 );
        EaValue.w = memory->GetWord( a[ EaReg ].l + Disp16 );
        ccc.Sm = WORD_MSB( EaValue.w );
        d[ Reg ].w += EaValue.w;

        pc += 2;    // modify PC
        break;
    case 6:     // Disp8(An, Xn).w
        Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
        Xn = GETBITS( pc + 2, 0x7000, 12 );
        Wl = GETBITS( pc + 2, 0x800, 11 );

        if( !Wl )       // word index
        {
            EaValue.w = memory->GetWord( a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w );
            ccc.Sm = WORD_MSB( EaValue.w );
            d[ Reg ].w += EaValue.w;
```

```
    }
    else
    {
        EaValue.w = memory->GetWord( a[ EaReg ].l + Disp8 + d[ Xn ].l );
        ccc.Sm = WORD_MSB( EaValue.w );
        d[ Reg ].w += EaValue.w;
    }

    pc += 2;    // modify PC
    break;
case 7:
    switch( EaReg )
    {
        case 0:         // (www).W
            ccc.Sm = WORD_MSB( memory->GetWord( memory->GetWord( pc + 2 )));
            d[ Reg ].w += memory->GetWord( memory->GetWord( pc + 2 ));

            pc += 2; // modify PC
            break;
        case 1:         // (www).L
            ccc.Sm = WORD_MSB( memory->GetWord( memory->GetLongword( pc + 2 )));
            d[ Reg ].w += memory->GetWord( memory->GetLongword( pc + 2 ));

            pc += 4; // modify PC
            break;
        case 2:         // (d16, PC).w
            Disp16 = memory->GetWord( pc + 2 );
            EaValue.w = memory->GetWord( pc + Disp16 );

            ccc.Sm = WORD_MSB( EaValue.w );
            d[ Reg ].w += EaValue.w;

            pc += 2; // modify PC
            break;
        case 3:         // d8(PC, Xn)
            Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
            Xn = GETBITS( pc + 2, 0x7000, 12 );
            Wl = GETBITS( pc + 2, 0x800, 11 );

            if ( !Wl )   // word index
            {
                EaValue.w = memory->GetWord( pc + Disp8 + (signed short int)d[ Xn ].w );
                ccc.Sm = WORD_MSB( EaValue.w );
                d[ Reg ].w += EaValue.w;
            }
            else      // long index
            {
                EaValue.w = memory->GetWord( pc + Disp8 + d[ Xn ].l );
                ccc.Sm = WORD_MSB( EaValue.w );
                d[ Reg ].w += EaValue.w;
            }
```

```
            pc += 2; // modify PC
            break;
        case 4:          // #<data>
            ccc.Sm = WORD_MSB( memory->GetWord( pc + 2) );
            d[ Reg ].w += memory->GetWord( pc + 2);

            pc += 2; // modify PC
            break;
        case 5:
        case 6:
        case 7:
            addx(); return;
            break;
      }
      break;
    }

    ccc.Rm = WORD_MSB( d[ Reg ].w );
    ccr.z = d[ EaReg ].w == 0;
    break;

  case 2:      // LONGWORD (Source: Ea)
    ccc.Dm = LONG_MSB( d[ Reg ].l );

    switch( EaMode )
    {
    case 0:      // Dn.l
      ccc.Sm = LONG_MSB( d[ EaReg ].l );
      d[ Reg ].l += d[ EaReg ].l;
      break;
    case 1:      // An.l
      ccc.Sm = LONG_MSB( a[ EaReg ].l );
      d[ Reg ].l += a[ EaReg ].l;
      break;
    case 2:      // (An).l
    case 3:      // (An)+.l
    case 4:      // -(An).l
      if( EaMode == 4 )a[ EaReg ].l -= 4;

      EaValue.l = memory->GetLongword( a[ EaReg ].l );
      ccc.Sm = LONG_MSB( EaValue.l );

      d[ Reg ].l += EaValue.l;

      if( EaMode == 3 )a[ EaReg ].l += 4;
      break;
    case 5:      // (d16, An).l
      Disp16 = memory->GetWord( pc + 2 );
      EaValue.l = memory->GetLongword( a[ EaReg ].l + Disp16 );
      ccc.Sm = LONG_MSB( EaValue.l );
      d[ Reg ].l += EaValue.l;
```

```
    pc += 2; // modify PC
    break;
case 6:     // Disp8(An, Xn).l
    Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
    Xn = GETBITS( pc + 2, 0x7000, 12 );
    Wl = GETBITS( pc + 2, 0x800, 11 );

    if ( !Wl )   // word index
        EaValue.l = memory->GetLongword( a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w );
    else     // long index
        EaValue.l = memory->GetLongword( a[ EaReg ].l + Disp8 + d[ Xn ].l );

    ccc.Sm = LONG_MSB( EaValue.l );
    d[ Reg ].l += EaValue.l;

    pc += 2; // modify PC
    break;
case 7:
    switch( EaReg )
    {
        case 0:         // (www).W
            ccc.Sm = LONG_MSB( memory->GetLongword( memory->GetWord( pc + 2 )));
            d[ Reg ].l += memory->GetLongword( memory->GetWord( pc + 2 ));

            pc += 2; // modify PC
            break;
        case 1:         // (www).L
            ccc.Sm = LONG_MSB( memory->GetLongword( memory->GetLongword( pc + 2 )));
            d[ Reg ].l += memory->GetLongword( memory->GetLongword( pc + 2 ));

            pc += 4; // modify PC
            break;
        case 2:         // (d16, PC).l
            Disp16 = memory->GetWord( pc + 2 );
            EaValue.l = memory->GetLongword( pc + Disp16 );

            ccc.Sm = LONG_MSB( EaValue.l );
            d[ Reg ].l += EaValue.l;

            pc += 2; // modify PC
            break;
        case 3:         // d8(PC, Xn)
            Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
            Xn = GETBITS( pc + 2, 0x7000, 12 );
            Wl = GETBITS( pc + 2, 0x800, 11 );

            if ( !Wl )   // word index
            {
                EaValue.l = memory->GetLongword( pc + Disp8 + (signed short int)d[ Xn ].w );
                ccc.Sm = LONG_MSB( EaValue.l );
                d[ Reg ].l += EaValue.l;
            }
```

```
            else      // long index
            {
               EaValue.l = memory->GetLongword( pc + Disp8 + d[ Xn ].l );
               ccc.Sm = LONG_MSB( EaValue.l );
               d[ Reg ].l += EaValue.l;
            }

            pc += 2; // modify PC
            break;
         case 4:          // #<data>
            ccc.Sm = LONG_MSB( memory->GetLongword( pc + 2) );
            d[ Reg ].l += memory->GetLongword( pc + 2);

            pc += 4; // modify PC
            break;
         case 5:
         case 6:
         case 7:
            addx(); return;
            break;
      }
      break;
   }

   ccc.Rm = LONG_MSB( d[ Reg ].l );
   ccr.z = d[ Reg ].l == 0;
   break;

case 4:      // BYTE (Source: Dn)
   ccc.Sm = BYTE_MSB( d[ Reg ].b );

   switch( EaMode )
   {
   case 0:      // Dn.b (n/a for dest = Ea)
   case 1:      // An.b (n/a for dest = Ea)
      addx(); return;
      break;
   case 2:      // (An).b
   case 3:      // (An)+.b
   case 4:      // -(An).b
      if( EaMode == 4 )a[ EaReg ].l--;

      EaValue.b = memory->GetByte( a[ EaReg ].l );
      ccc.Dm = BYTE_MSB( EaValue.b );
      Result.b = d[ Reg ].b + EaValue.b;

      memory->SetByte( Result.b, a[ EaReg ].l );

      if( EaMode == 3 )a[ EaReg ].l++;
      break;
   case 5:      // (d16, An).b
      Disp16 = memory->GetWord( pc + 2 );
```

```
    DispedMem = a[ EaReg ].l + Disp16;
    EaValue.b = memory->GetByte( DispedMem );
    ccc.Dm = BYTE_MSB( EaValue.b );
    Result.b = d[ Reg ].b + EaValue.b;

    memory->SetByte( Result.b, DispedMem );

    pc += 2; // modify PC
    break;
case 6:     // Disp8(An, Xn).b

    Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
    Xn = GETBITS( pc + 2, 0x7000, 12 );
    Wl = GETBITS( pc + 2, 0x800, 11 );

    if ( !Wl )   // word index
       DispedMem = a[ EaReg ].l + (unsigned long)Disp8 + (unsigned long)(signed short int)d[ Xn ].
            w ;
    else     // long index
       DispedMem = a[ EaReg ].l + (unsigned long)Disp8 + d[ Xn ].l;

    EaValue.b = memory->GetByte( DispedMem );

    ccc.Dm = BYTE_MSB( EaValue.b );
    Result.b = d[ Reg ].b + EaValue.b;

    memory->SetByte( Result.b, DispedMem );

    pc += 2; // modify PC
    break;
case 7:
    switch( EaReg )
    {
       case 0:          // (www).W
          ccc.Dm = BYTE_MSB( memory->GetByte( memory->GetWord( pc + 2 )));
          Result.b = d[ Reg ].b + memory->GetByte( memory->GetWord( pc + 2 ));

          memory->SetByte( Result.b, memory->GetWord( pc + 2 ));

          pc += 2; // modify PC
          break;
       case 1:          // (www).L
          ccc.Dm = BYTE_MSB( memory->GetByte( memory->GetLongword( pc + 2 )));
          Result.b = d[ Reg ].b + memory->GetByte( memory->GetLongword( pc + 2 ));

          memory->SetByte( Result.b, memory->GetLongword( pc + 2 ));

          pc += 4; // modify PC
          break;
       case 2:          // (d16, PC).b
       case 3:          // d8(PC, Xn)
```

```
            case 4:          // #<data>
            case 5:
            case 6:
            case 7:
               addx(); return;
               break;
         }
         break;
      }

      ccc.Rm = BYTE_MSB( Result.b );
      ccr.z = Result.b == 0;
      break;

   case 5:      // WORD (Source: Dn)
      ccc.Sm = WORD_MSB( d[ Reg ].w );

      switch( EaMode )
      {
      case 0:      // Dn.w (n/a for dest = Ea)
      case 1:      // An.w (n/a for dest = Ea)
         addx(); return;
         break;
      case 2:      // (An).w
      case 3:      // (An)+.w
      case 4:      // -(An).w
         if( EaMode == 4 )a[ EaReg ].l -= 2;

         ccc.Dm = WORD_MSB( memory->GetWord( a[ EaReg ].l ));
         Result.w = d[ Reg ].w + memory->GetWord( a[ EaReg ].l );

         memory->SetWord( Result.w, a[ EaReg ].l );

         if( EaMode == 3 )a[ EaReg ].l += 2;
         break;
      case 5:      // (d16, An).w
         Disp16 = memory->GetWord( pc + 2 );

         DispedMem = a[ EaReg ].l + Disp16;
         EaValue.w = memory->GetWord( DispedMem );

         ccc.Dm = WORD_MSB( EaValue.w );

         Result.w = d[ Reg ].w + EaValue.w;

         memory->SetWord( Result.w, DispedMem );
         pc += 2; // modify PC
         break;
      case 6:      // Disp8(An, Xn).w
         Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
         Xn = GETBITS( pc + 2, 0x7000, 12 );
         Wl = GETBITS( pc + 2, 0x800, 11 );
```

```
            if ( !Wl )    // word index
               DispedMem = a[ EaReg ].l + (unsigned long)Disp8 + (unsigned long)(signed short int)d[ Xn ].
                    w ;
            else      // long index
               DispedMem = a[ EaReg ].l + (unsigned long)Disp8 + d[ Xn ].l;

            EaValue.w = memory->GetWord( DispedMem );

            ccc.Dm = WORD_MSB( EaValue.w );
            Result.w = d[ Reg ].w + EaValue.w;

            memory->SetByte( Result.b, DispedMem );

            pc += 2; // modify PC
            break;
         case 7:
            switch( EaReg )
            {
               case 0:          // (www).W
                  ccc.Dm = memory->GetWord( memory->GetWord( pc + 2 ));
                  Result.w = d[ Reg ].w + memory->GetWord( memory->GetWord( pc + 2 ));

                  memory->SetWord( Result.w, memory->GetWord( pc + 2 ));

                  pc += 2; // modify PC
                  break;
               case 1:          // (www).L
                  ccc.Dm = memory->GetWord( memory->GetLongword( pc + 2 ));
                  Result.w = d[ Reg ].w + memory->GetWord( memory->GetLongword( pc + 2 ));

                  memory->SetWord( Result.w, memory->GetLongword( pc + 2 ));

                  pc += 4; // modify PC
                  break;
               case 2:          // (d16, PC).w (n/a for dest = Ea)
               case 3:          // d8(PC, Xn) (n/a for dest = Ea)
               case 4:          // #<data> (n/a for dest = Ea)
               case 5:
               case 6:
               case 7:
                  addx(); return;
                  break;
            }
            break;
         }

         ccc.Rm = WORD_MSB( Result.w );
         ccr.z = Result.w == 0;
         break;

      case 6:     // LONGWORD (Source: Dn)
```

```
ccc.Sm = LONG_MSB( d[ Reg ].l );

switch( EaMode )
{
case 0:     // Dn.l (n/a for dest = Ea)
case 1:     // An.l (n/a for dest = Ea)
   addx(); return;
   break;
case 2:     // (An).l
case 3:     // (An)+.l
case 4:     // -(An).l
   if( EaMode == 4 )a[ EaReg ].l -= 4;
   EaValue.l = memory->GetLongword( a[ EaReg ].l );

   ccc.Dm = LONG_MSB( EaValue.l );
   Result.l = d[ Reg ].l + EaValue.l;

   memory->SetLongword( Result.l, a[ EaReg ].l );

   if( EaMode == 3 )a[ EaReg ].l += 4;
   break;
case 5:     // (d16, An).l
   Disp16 = memory->GetWord( pc + 2 );

   DispedMem = a[ EaReg ].l + Disp16;
   EaValue.l = memory->GetLongword( DispedMem );

   ccc.Dm = LONG_MSB( EaValue.w );
   Result.l = d[ Reg ].l + EaValue.l;

   memory->SetLongword( Result.l, DispedMem );

   pc += 2; // modify PC
   break;
case 6:     // Disp8(An, Xn).l
   Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
   Xn = GETBITS( pc + 2, 0x7000, 12 );
   Wl = GETBITS( pc + 2, 0x800, 11 );

   if ( !Wl )   // word index
      DispedMem = a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w;
   else     // long index
      DispedMem = a[ EaReg ].l + Disp8 + d[ Xn ].l;

   EaValue.l = memory->GetLongword( DispedMem );
   ccc.Dm = LONG_MSB( EaValue.l );
   Result.l = d[ Reg ].l + EaValue.l;
   memory->SetLongword( Result.l, DispedMem );

   pc += 2; // modify PC
   break;
```

```
        case 7:
            switch( EaReg )
            {
                case 0:         // (www).W
                    ccc.Dm = LONG_MSB( memory->GetLongword( memory->GetWord( pc + 2 )));
                    Result.l = d[ Reg ].l + memory->GetLongword( memory->GetWord( pc + 2 ));

                    memory->SetLongword( Result.l, memory->GetWord( pc + 2 ));

                    pc += 2; // modify PC
                    break;
                case 1:         // (www).L
                    ccc.Dm = LONG_MSB( memory->GetLongword( memory->GetLongword( pc + 2 )));
                    Result.l = d[ Reg ].l + memory->GetLongword( memory->GetLongword( pc + 2 ));

                    memory->SetLongword( Result.l, memory->GetLongword( pc + 2 ));

                    pc += 4; // modify PC
                    break;
                case 2:         // (d16, PC).l (n/a for dest = Ea)
                case 3:         // d8(PC, Xn) (n/a for dest = Ea)
                case 4:         // #<data> (n/a for dest = Ea)
                case 5:
                case 6:
                case 7:
                    addx(); return;
                    break;
            }
            break;
        }
        ccc.Rm = LONG_MSB( Result.l );
        ccr.z = Result.l == 0;
        break;
    case 3:
        //
        // ADDA: Add to address register
        //

        // WORD

        ccc.Dm = WORD_MSB( a[ Reg ].w );

        switch( EaMode )
        {
        case 0:     // Dn.w
            ccc.Sm = WORD_MSB( d[ EaReg ].w );
            a[ Reg ].w += d[ EaReg ].w;
            break;
        case 1:     // An.w
            ccc.Sm = WORD_MSB( a[ EaReg ].w );
            a[ Reg ].w += a[ EaReg ].w;
            break;
```

```
case 2:     // (An).w
case 3:     // (An)+.w
case 4:     // -(An).w
    if( EaMode == 4 )a[ EaReg ].l -= 2;

    EaValue.w = memory->GetWord( a[ EaReg ].l);
    ccc.Sm = WORD_MSB( EaValue.w );

    a[ Reg ].w += EaValue.w;

    if( EaMode == 3 )a[ EaReg ].l += 2;
    break;
case 5:     // (d16, An).w
    Disp16 = memory->GetWord( pc + 2 );
    EaValue.w = memory->GetWord( a[ EaReg ].l + Disp16 );

    ccc.Sm = WORD_MSB( EaValue.w );

    a[ Reg ].w += EaValue.w;

    pc += 2; // modify PC
    break;
case 6:     // Disp8(An, Xn).w
    Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
    Xn = GETBITS( pc + 2, 0x7000, 12 );
    Wl = GETBITS( pc + 2, 0x800, 11 );

    if ( !Wl )   // word index
        EaValue.w = memory->GetWord( a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w );

    else     // long index
        EaValue.w = memory->GetWord( a[ EaReg ].l + Disp8 + d[ Xn ].l );

    ccc.Sm = WORD_MSB( EaValue.w );
    a[ Reg ].w += EaValue.w;

    pc += 2; // modify PC
    break;
case 7:
    switch( EaReg )
    {
        case 0:         // (www).W
            ccc.Sm = WORD_MSB( memory->GetWord( memory->GetWord( pc + 2 )));
            a[ Reg ].w += memory->GetWord( memory->GetWord( pc + 2 ));

            pc += 2; // modify PC
            break;
        case 1:         // (www).L
            ccc.Sm = WORD_MSB( memory->GetWord( memory->GetLongword( pc + 2 )));
            a[ Reg ].w += memory->GetWord( memory->GetLongword( pc + 2 ));

            pc += 4; // modify PC
```

```
            break;
        case 2:          // (d16, PC).w
            Disp16 = memory->GetWord( pc + 2 );
            ccc.Sm = WORD_MSB( memory->GetWord( pc + Disp16 ));
            a[ Reg ].w += memory->GetWord( pc + Disp16 );

            pc += 2; // modify PC
            break;
        case 3:          // d8(PC, Xn)
            Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
            Xn = GETBITS( pc + 2, 0x7000, 12 );
            Wl = GETBITS( pc + 2, 0x800, 11 );

            if ( !Wl )   // word index
                EaValue.w = memory->GetWord( pc + Disp8 + (signed short int)d[ Xn ].w );
            else      // long index
                EaValue.w = memory->GetWord( pc + Disp8 + d[ Xn ].l );

            ccc.Sm = WORD_MSB( EaValue.w );
            a[ Reg ].w += EaValue.w;

            pc += 2; // modify PC
            break;
        case 4:          // #<data>
            ccc.Sm = WORD_MSB( memory->GetWord( pc + 2) );
            a[ Reg ].w += memory->GetWord( pc + 2);

            pc += 2; // modify PC
            break;
        case 5:
        case 6:
        case 7:
            addx(); return;
            break;
    }
    break;
}
ccc.Rm = WORD_MSB( a[ Reg ].w );
ccr.z = a[ Reg ].w == 0;
break;

case 7:
    ccc.Dm = LONG_MSB( a[ Reg ].l );

    switch( EaMode )
    {
    // LONGWORD (Source: Ea)
    case 0:      // Dn.l
        ccc.Sm = LONG_MSB( d[ EaReg ].l );
        a[ Reg ].l += d[ EaReg ].l;
        break;
    case 1:      // An.l
```

```
    ccc.Sm = LONG_MSB( a[ EaReg ].l );
    a[ Reg ].l += a[ EaReg ].l;
    break;
case 2:     // (An).l
case 3:     // (An)+.l
case 4:     // -(An).l
    if( EaMode == 4 )a[ EaReg ].l -= 4;
    EaValue.l = memory->GetLongword( a[ EaReg ].l );

    ccc.Sm = LONG_MSB( EaValue.l );
    a[ Reg ].l += EaValue.l;

    if( EaMode == 3) a[ EaReg ].l += 4;
    break;
case 5:     // (d16, An).l
    Disp16 = memory->GetWord( pc + 2 );
    EaValue.l = memory->GetLongword( a[ EaReg ].l + Disp16 );
    ccc.Sm = LONG_MSB( EaValue.l );

    a[ Reg ].l += EaValue.l;

    pc += 2; // modify PC
    break;
case 6:     // Disp8(An, Xn).l
    Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
    Xn = GETBITS( pc + 2, 0x7000, 12 );
    Wl = GETBITS( pc + 2, 0x800, 11 );

    if ( !Wl )   // word index
        EaValue.l = memory->GetLongword( a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w );
    else     // long index
        EaValue.l = memory->GetLongword( a[ EaReg ].l + Disp8 + d[ Xn ].l );

    ccc.Sm = LONG_MSB( EaValue.l );
    a[ Reg ].l += EaValue.l;

    pc += 2; // modify PC
    break;
case 7:
    switch( EaReg )
    {
        case 0:         // (www).W
            ccc.Sm = LONG_MSB( memory->GetLongword( memory->GetWord( pc + 2 )));
            a[ Reg ].l += memory->GetLongword( memory->GetWord( pc + 2 ));

            pc += 2; // modify PC
            break;
        case 1:         // (www).L
            ccc.Sm = LONG_MSB( memory->GetLongword( memory->GetLongword( pc + 2 )));
            a[ Reg ].l += memory->GetLongword( memory->GetLongword( pc + 2 ));

            pc += 4; // modify PC
```

```
                break;
            case 2:          // (d16, PC).l
                Disp16 = memory->GetWord( pc + 2 );
                ccc.Sm = LONG_MSB( memory->GetLongword( pc + Disp16 ));

                a[ Reg ].l += memory->GetLongword( pc + Disp16 );

                pc += 2; // modify PC
                break;
            case 3:          // d8(PC, Xn)
                Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
                Xn = GETBITS( pc + 2, 0x7000, 12 );
                Wl = GETBITS( pc + 2, 0x800, 11 );

                if ( !Wl )   // word index
                    EaValue.l = memory->GetLongword( pc + Disp8 + (signed short int)d[ Xn ].w );
                else      // long index
                    EaValue.l = memory->GetLongword( pc + Disp8 + d[ Xn ].l );

                ccc.Sm = LONG_MSB( EaValue.l );
                a[ Reg ].l += EaValue.l;

                pc += 2; // modify PC
                break;
            case 4:          // #<data>
                ccc.Sm = LONG_MSB( memory->GetLongword( pc + 2) );
                a[ Reg ].l += memory->GetLongword( pc + 2);

                pc += 4; // modify PC
                break;
            case 5:
            case 6:
            case 7:
                addx(); return;
                break;
        }
        break;
    }
    ccc.Rm = LONG_MSB( a[ Reg ].l );
    ccr.z = a[ Reg ].l == 0;
    break;
  }


  // Set remaining CCR flags (pp. 89 Motorola Manual)
  ccr.c = ccr.x = ccc.Sm & ccc.Dm | ~ccc.Rm & ccc.Dm | ccc.Sm & ~ccc.Rm;
  ccr.v = ccc.Sm & ccc.Dm & ~ccc.Rm | !ccc.Sm & ~ccc.Dm & ccc.Rm;
  ccr.n = ccc.Rm;

  // Update PC
  pc += 2;
}
```

```
void M68008::addi()
{
   //
   // Alan Donnelly and Gerard Whyte (27/02/02)
   //

   unsigned char Size = GET_INSTRBITS(0xC0, 6);
   unsigned char EaMode = GET_INSTRBITS(0x38, 3);
   unsigned char EaReg = GET_INSTRBITS(0x7, 0);

   unsigned long DispedMem = 0;              // Displaced memory address
   DATA_REGISTER( Result );                  // Result of addition
   DATA_REGISTER( Data );                    // Immediate data
   CCC ccc;                                  // Condition code calculations

   // (d16, An)
   signed short int Disp16;

   // Disp8(An, Xn)
   signed char Disp8;
   unsigned char Xn;
   unsigned char Wl;

   switch( Size )
   {
      case 0:      // BYTE
         Data.b = memory->GetWord( pc + 2 )& 0xFF;
         ccc.Sm = BYTE_MSB( Data.b );

         pc += 2;
         switch( EaMode )
         {
            case 0:      // Dn.b
               ccc.Dm = BYTE_MSB( d[ EaReg ].b );
               Result.b = d[ EaReg ].b + Data.b;
               d[ EaReg ].b = Result.b;

               break;
            case 1:      // An.b (n/a)
               illegal ();
               break;
            case 2:      // (An).b
            case 3:      // (An)++.b
            case 4:      // --(An).b
               if ( EaMode == 4 )a[ EaReg ].l--;

               ccc.Dm = BYTE_MSB( memory->GetByte( a[ EaReg ].l ));
               Result.b = memory->GetByte( a[ EaReg ].l )+ Data.b;
               memory->SetByte( Result.b, a[ EaReg ].l );

               if ( EaMode == 3) a[ EaReg ].l++;
```

```
    break;
case 5:     // Disp16(An).b
    Disp16 = memory->GetWord( pc + 2 );

    DispedMem = a[ EaReg ].l + Disp16;
    ccc.Dm = BYTE_MSB( memory->GetByte( DispedMem ));
    Result.b = memory->GetByte( DispedMem )+ Data.b;
    memory->SetByte( Result.b, DispedMem );

    // Update PC
    pc += 2;
    break;
case 6:     // Disp8(An, Xn).b
    Disp8 = memory->GetByte( pc + 3 );
    Xn = GETBITS( pc + 2, 0x7000, 12 );
    Wl = GETBITS( pc + 2, 0x800, 11 );

    if ( ! Wl )
        DispedMem = a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w;
    else
        DispedMem = a[ EaReg ].l + Disp8 + d[ Xn ].l;

    ccc.Dm = BYTE_MSB( memory->GetByte( DispedMem ));
    Result.b = memory->GetByte( DispedMem )+ Data.b;
    memory->SetByte( Result.b, DispedMem );

    // Update PC
    pc += 2;
    break;
case 7:
    switch ( EaReg )
    {
        case 0: // (xxx).W.b
            ccc.Dm = BYTE_MSB( memory->GetByte( memory->GetWord( pc + 2 )));
            Result.b = Data.b + memory->GetByte( memory->GetWord( pc + 2 ));
            memory->SetByte( Result.b, memory->GetWord( pc + 2 ));

            // Update PC
            pc += 2;
            break;
        case 1: // (xxx).L.b
            ccc.Dm = BYTE_MSB( memory->GetByte( memory->GetLongword( pc + 2 )));
            Result.b = Data.b + memory->GetByte( memory->GetLongword( pc + 2 ));
            memory->SetByte( Result.b, memory->GetLongword( pc + 2 ));

            // Update PC
            pc += 4;
            break;
        default:
            illegal ();
            break;
    }
```

```
            break;
        }

        ccc.Rm = BYTE_MSB( Result.b );
        ccr.z = Result.b == 0;
        break;

    case 1:     // WORD
        Data.w = memory->GetWord( pc + 2 )& 0xFFFF;
        ccc.Sm = WORD_MSB( Data.w );
        pc += 2;
        switch( EaMode )
        {
            case 0:     // Dn.w
                ccc.Dm = WORD_MSB( d[ EaReg ].w );
                Result.w = d[ EaReg ].w + Data.w;
                d[ EaReg ].w = Result.w;
                break;
            case 1:     // An.w
                ccc.Dm = WORD_MSB( a[ EaReg ].w );
                Result.w = a[ EaReg ].w + Data.w;
                a[ EaReg ].w += Result.w;
                break;
            case 2:     // (An).w
            case 3:     // (An)++.w
            case 4:     // --(An).w
                if( EaMode == 4 )a[ EaReg ].l -= 2;

                ccc.Dm = WORD_MSB( memory->GetWord( a[ EaReg ].l ));
                Result.w = memory->GetWord( a[ EaReg ].l )+ Data.w;
                memory->SetWord( Result.w, a[ EaReg ].l );

                if( EaMode == 3) a[ EaReg ].l += 2;
                break;
            case 5:     // Disp16(An).w
                Disp16 = memory->GetWord( pc + 2 );

                DispedMem = a[ EaReg ].l + Disp16;
                ccc.Dm = WORD_MSB( memory->GetWord( DispedMem ));
                Result.w = memory->GetWord( DispedMem )+ Data.w;

                memory->SetWord( Result.w, DispedMem );

                // Update PC
                pc += 2;
                break;
            case 6:     // Disp8(An, Xn).w
                Disp8 = memory->GetByte( pc + 3 );
                Xn = GETBITS( pc + 2, 0x7000, 12 );
                Wl = GETBITS( pc + 2, 0x800, 11 );

                if  ( ! Wl )
```

```
            DispedMem = a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w;
         else
            DispedMem = a[ EaReg ].l + Disp8 + d[ Xn ].l;

         ccc.Dm = WORD_MSB( memory->GetWord( DispedMem ));
         Result.w = memory->GetWord( DispedMem )+ Data.w;

         memory->SetWord( Result.w, DispedMem );

         // Update PC
         pc += 2;
         break;
      case 7:
         switch ( EaReg )
         {
            case 0: // (xxx).W.w
               ccc.Dm = WORD_MSB( memory->GetWord( memory->GetWord( pc + 2 )));
               Result.w = Data.w + memory->GetWord( memory->GetWord( pc + 2 ));

               memory->SetWord( Result.w, memory->GetWord( pc + 2 ));

               // Update PC
               pc += 2;
               break;
            case 1: // (xxx).L.w
               ccc.Dm = WORD_MSB( memory->GetWord( memory->GetLongword( pc + 2 )));
               Result.w = Data.w + memory->GetWord( memory->GetLongword( pc + 2 ));

               memory->SetWord( Result.w, memory->GetLongword( pc + 2 ));

               // Update PC
               pc += 4;
               break;
            default:
               illegal ();
               break;
         }
         break;
   }
   ccc.Rm = WORD_MSB( Result.w );
   ccr.z = Result.w == 0;
   break;

case 2: // LONGWORD
   Data.l = memory->GetLongword( pc + 2 );
   pc += 4;

   switch( EaMode )
   {
      case 0:      // Dn.l
         ccc.Dm = LONG_MSB( d[ EaReg ].l );
         Result.l = d[ EaReg ].l + Data.l;
```

```
        d[ EaReg ].l = Result.l;
        break;
    case 1:     // An.l
        ccc.Dm = LONG_MSB( a[ EaReg ].l );
        Result.l = a[ EaReg ].l + Data.l;
        a[ EaReg ].l = Result.l;
        break;
    case 2:     // (An).l
    case 3:     // (An)++.l
    case 4:     // --(An).l
        if( EaMode == 4 )a[ EaReg ].l -= 4;

        ccc.Dm = LONG_MSB( memory->GetLongword( a[ EaReg ].l ));
        Result.l = memory->GetLongword( a[ EaReg ].l )+ Data.l;
        memory->SetLongword( Result.l, a[ EaReg ].l );

        if( EaMode == 3) a[ EaReg ].l += 4;
        break;
    case 5:     // Disp16(An).l
        Disp16 = memory->GetWord( pc + 2 );
        DispedMem = a[ EaReg ].l + Disp16;

        ccc.Dm = LONG_MSB( memory->GetLongword( DispedMem ));
        Result.l = memory->GetLongword( DispedMem )+ Data.l;

        memory->SetLongword( Result.l, DispedMem );

        // Update PC
        pc += 2;
        break;
    case 6:     // Disp8(An, Xn).l
        Disp8 = memory->GetByte( pc + 3 );
        Xn = GETBITS( pc + 2, 0x7000, 12 );
        Wl = GETBITS( pc + 2, 0x800, 11 );

        if  ( ! Wl )
            DispedMem = a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w;
        else
            DispedMem = a[ EaReg ].l + Disp8 + d[ Xn ].l;

        ccc.Dm = LONG_MSB( memory->GetLongword( DispedMem ));
        Result.l = memory->GetLongword( DispedMem )+ Data.l;
        memory->SetLongword( Result.l, DispedMem );

        // Update PC
        pc += 2;
        break;
    case 7:
        switch ( EaReg )
        {
            case 0: // (xxx).W.l
                ccc.Dm = LONG_MSB( memory->GetLongword( memory->GetWord( pc + 2 )));
```

```
                    Result.l = Data.l + memory->GetLongword( memory->GetWord( pc + 2 ));
                    memory->SetLongword( Result.l, memory->GetWord( pc + 2 ));

                    // Update PC
                    pc += 2;
                    break;
                case 1: // (xxx).L.l
                    ccc.Dm = LONG_MSB( memory->GetLongword( memory->GetLongword( pc + 2 )));
                    Result.l = Data.l + memory->GetLongword( memory->GetLongword( pc + 2 ));
                    memory->SetLongword( Result.l, memory->GetLongword( pc + 2 ));

                    // Update PC
                    pc += 4;
                    break;
                default:
                    illegal ();
                    break;
            }
            break;
        }
        ccc.Rm = LONG_MSB( Result.l );
        ccr.z = Result.l == 0;
        break;
    }

    // Set remaining CCR flags
    ccr.c = ccr.x = ccc.Sm & ccc.Dm | ~ccc.Rm & ccc.Dm | ccc.Sm & ~ccc.Rm;
    ccr.v = ccc.Sm & ccc.Dm & ~ccc.Rm | !ccc.Sm & ~ccc.Dm & ccc.Rm;
    ccr.n = ccc.Rm;

    // Update PC
    pc += 2;
}

void M68008::addq()
{
    //
    // Alan Donnelly and Gerard Whyte (27/02/02)
    //

    unsigned char Data = GET_INSTRBITS(0xE00, 9);
    unsigned char Size = GET_INSTRBITS(0xC0, 6);
    unsigned char EaMode = GET_INSTRBITS(0x38, 3);
    unsigned char EaReg = GET_INSTRBITS(0x7, 0);

    unsigned long DispedMem = 0;            // Displaced memory address
    DATA_REGISTER( Result );                // Result of addition
    CCC ccc;                                // Condition code calculations

    // (d16, An)
    signed short int Disp16;
```

```
// Disp8(An, Xn)
signed char Disp8;
unsigned char Xn;
unsigned char Wl;

// When Data == 0, we addq 8
if( Data == 0 )Data = 8;

switch ( Size )
{

// BYTE (Source: Dn)
case 0:
    ccc.Sm = BYTE_MSB( Data );

    switch( EaMode )
    {
    case 0:      // Dn.b
        ccc.Dm = BYTE_MSB( d[ EaReg ].b );
        d[ EaReg ].b += Data;
        Result.b = d[ EaReg ].b;
        break;
    case 1:      // An.b (n/a for dest = Ea)
         illegal ();
        break;
    case 2:      // (An).b
    case 3:      // (An)+.b
    case 4:      // -(An).b
        if( EaMode == 4 )a[ EaReg ].l--;

        ccc.Dm = BYTE_MSB( memory->GetByte( a[ EaReg ].l ));
        Result.b = memory->GetByte( a[ EaReg ].l )+ Data;
        memory->SetByte( Result.b, a[ EaReg ].l );

        if( EaMode == 3) a[ EaReg ].l++;
        break;
    case 5:      // (d16, An).b
        Disp16 = memory->GetWord( pc + 2 );

        DispedMem = a[ EaReg ].l + Disp16;
        ccc.Dm = BYTE_MSB( memory->GetByte( DispedMem ));

        Result.b = Data + memory->GetByte( DispedMem );
        memory->SetByte( Result.b, DispedMem );

        // Update PC
        pc += 2;
        break;
    case 6:      // Disp8(An, Xn).b
        Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
        Xn = GETBITS( pc + 2, 0x7000, 12 );
        Wl = GETBITS( pc + 2, 0x800, 11 );
```

```
            if ( !Wl )        // word index
               DispedMem = a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w;
            else
               DispedMem = a[ EaReg ].l + Disp8 + d[ Xn ].l;

            ccc.Dm = BYTE_MSB( memory->GetByte( DispedMem ));
            Result.b = Data + memory->GetByte( DispedMem );
            memory->SetByte( Result.b, DispedMem );

            // Update PC
            pc += 2;
            break;
         case 7:
            switch( EaReg )
            {
               case 0:          // (www).W
                  ccc.Dm = BYTE_MSB( memory->GetByte( memory->GetWord( pc + 2 )));
                  Result.b = Data + memory->GetByte( memory->GetWord( pc + 2 ));
                  memory->SetByte( Result.b, memory->GetWord( pc + 2 ));

                  // Update PC
                  pc += 2;
                  break;
               case 1:          // (www).L
                  ccc.Dm = BYTE_MSB( memory->GetByte( memory->GetLongword( pc + 2 )));
                  Result.b = Data + memory->GetByte( memory->GetLongword( pc + 2 ));
                  memory->SetByte( Result.b, memory->GetLongword( pc + 2 ));

                  // Update PC
                  pc += 4;
                  break;
               case 2:          // (d16, PC).b (n/a)
               case 3:          // d8(PC, Xn) (n/a)
               case 4:          // #<data> (n/a)
               case 5:
               case 6:
               case 7:
                   illegal ();
                  break;
            }
            break;
         }
         ccc.Rm = BYTE_MSB( Result.b );
         ccr.z = Result.b == 0;
         break;

      // WORD
      case 1:
         ccc.Sm = WORD_MSB( Data );

         switch( EaMode )
```

```
{
case 0:      // Dn.w
   ccc.Dm = WORD_MSB( d[ EaReg ].l );
   d[ EaReg ].w += Data;
   Result.w = d[ EaReg ].w;
   break;
case 1:      // An.w
   ccc.Dm = WORD_MSB( a[ EaReg ].l );
   a[ EaReg ].w += Data;
   Result.w = a[ EaReg ].w;
   break;
case 2:      // (An).w
case 3:      // (An)+.w
case 4:      // -(An).w
   if( EaMode == 4 )a[ EaReg ].l -= 2;

   ccc.Dm = WORD_MSB( memory->GetWord( a[ EaReg ].l ));
   Result.w = memory->GetWord( a[ EaReg ].l )+ Data;
   memory->SetWord( Result.w, a[ EaReg ].l );

   if( EaMode == 3) a[ EaReg ].l += 2;
   break;
case 5:      // (d16, An).w
   Disp16 = memory->GetWord( pc + 2 );
   DispedMem = a[ EaReg ].l + Disp16;

   ccc.Dm = WORD_MSB( memory->GetWord( DispedMem ));
   Result.w = memory->GetWord( DispedMem )+ Data;
   memory->SetWord( Result.w, DispedMem );

   // Update PC
   pc += 2;
   break;
case 6:      // Disp8(An, Xn).w
   Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
   Xn = GETBITS( pc + 2, 0x7000, 12 );
   Wl = GETBITS( pc + 2, 0x800, 11 );

   if ( !Wl )       // word index
      DispedMem = a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w;
   else
      DispedMem = a[ EaReg ].l + Disp8 + d[ Xn ].l;

   ccc.Dm = WORD_MSB( memory->GetWord( DispedMem ));
   Result.w = Data + memory->GetWord( DispedMem );
   memory->SetWord( Result.w, DispedMem );

   // Update PC
   pc += 2;
   break;
case 7:
   switch( EaReg )
```

```
        {
            case 0:         // (www).W
                ccc.Dm = WORD_MSB( memory->GetWord( memory->GetWord( pc + 2 )));
                Result.w = Data + memory->GetWord( memory->GetWord( pc + 2 ));
                memory->SetWord( Result.w, memory->GetWord( pc + 2 ));

                // Update PC
                pc += 2;
                break;
            case 1:         // (www).L
                ccc.Dm = WORD_MSB( memory->GetWord( memory->GetLongword( pc + 2 )));
                Result.w = Data + memory->GetWord( memory->GetLongword( pc + 2 ));
                memory->SetWord( Result.w, memory->GetLongword( pc + 2 ));

                // Update PC
                pc += 4;
                break;
            case 2:         // (d16, PC).w (n/a)
            case 3:         // d8(PC, Xn) (n/a)
            case 4:         // #<data> (n/a)
            case 5:
            case 6:
            case 7:
                illegal ();
                break;
        }
        break;
    }

    ccc.Rm = WORD_MSB( Result.w );
    ccr.x = Result.w == 0;
    break;

// LONGWORD (Source: Dn)
case 2:
    ccc.Sm = LONG_MSB( Data );

    switch( EaMode )
    {
    case 0:     // Dn.l
        ccc.Dm = LONG_MSB( d[ EaReg ].l );
        d[ EaReg ].l += Data;
        Result.l = d[ EaReg ].l;
        break;
    case 1:     // An.l (n/a for dest = Ea)
        a[ EaReg ].l += Data;
        break;
    case 2:     // (An).l
    case 3:     // (An)+.l
    case 4:     // -(An).l
        if( EaMode == 4 )a[ EaReg ].l -= 4;
```

```
    ccc.Dm = LONG_MSB( memory->GetLongword( a[ EaReg ].l ));
    Result.l = memory->GetLongword( a[ EaReg ].l )+ Data;
    memory->SetLongword( Result.l, a[ EaReg ].l );

    if ( EaMode == 3) a[ EaReg ].l += 4;
    break;
  case 5:      // (d16, An).l
    Disp16 = memory->GetWord( pc + 2 );
    DispedMem = a[ EaReg ].l + Disp16;

    ccc.Dm = LONG_MSB( memory->GetLongword( DispedMem ));
    Result.l = Data + memory->GetLongword( DispedMem );
    memory->SetLongword( Result.l, DispedMem );

    // Update PC
    pc += 2;
    break;
  case 6:      // Disp8(An, Xn).l
    Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
    Xn = GETBITS( pc + 2, 0x7000, 12 );
    Wl = GETBITS( pc + 2, 0x800, 11 );

    if ( !Wl )       // word index
      DispedMem = a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w;
    else
      DispedMem = a[ EaReg ].l + Disp8 + d[ Xn ].l;

    ccc.Dm = LONG_MSB( memory->GetLongword( DispedMem ));
    Result.l = Data + memory->GetLongword( DispedMem );
    memory->SetLongword( Result.l, DispedMem );

    // Update PC
    pc += 2;
    break;
  case 7:
    switch( EaReg )
    {
      case 0:          // (www).W
        ccc.Dm = LONG_MSB( memory->GetLongword( memory->GetWord( pc + 2 )));
        Result.l = Data + memory->GetLongword( memory->GetWord( pc + 2 ));
        memory->SetLongword( Result.l, memory->GetWord( pc + 2 ));

        // Update PC
        pc += 2;
        break;
      case 1:          // (www).L
        ccc.Dm = LONG_MSB( memory->GetLongword( memory->GetLongword( pc + 2 )));
        Result.l = Data + memory->GetLongword( memory->GetLongword( pc + 2 ));
        memory->SetLongword( Result.l, memory->GetLongword( pc + 2 ));

        // Update PC
        pc += 4;
```

```
                break;
            case 2:         // (d16, PC).l (n/a)
            case 3:         // d8(PC, Xn) (n/a)
            case 4:         // #<data> (n/a)
            case 5:
            case 6:
            case 7:
                illegal();
                break;
            }
            break;
        }

        ccc.Rm = LONG_MSB( Result.l );
        ccr.z = Result.l == 0;
    }

    // Set remaining CCR flags
    ccr.c = ccr.x = ccc.Sm & ccc.Dm | ~ccc.Rm & ccc.Dm | ccc.Sm & ~ccc.Rm;
    ccr.v = ccc.Sm & ccc.Dm & ~ccc.Rm | !ccc.Sm & ~ccc.Dm & ccc.Rm;
    ccr.n = ccc.Rm;

    // Update PC
    pc += 2;

}

void M68008::addx()
{
    unsigned char Size = GET_INSTRBITS(0xC0, 6);
    unsigned char Rx = GET_INSTRBITS(0xE00, 9);
    unsigned char Ry = GET_INSTRBITS(0x7, 0);
    unsigned char Rm = GET_INSTRBITS(0x8, 3);

    CCC ccc;                                // Condition code calculations

    switch( Size )
    {
        case 0:     // BYTE
            if ( !Rm )  // Data Reg. to Data Reg.
            {
                ccc.Sm = BYTE_MSB( d[ Ry ].b );
                ccc.Dm = BYTE_MSB( d[ Rx ].b );

                d[ Rx ].b += d[ Ry ].b + ccr.x;

                ccc.Rm = BYTE_MSB( d[ Rx ].b );
                ccr.z = d[ Rx ].b == 0;
            }
            else    // Address to Address
            {
                a[ Rx ].l--;
```

```
            a[ Ry ].l--;
            ccc.Sm = BYTE_MSB( memory->GetByte( a[ Ry ].l )); // + ccr.x; ??
            ccc.Dm = BYTE_MSB( memory->GetByte( a[ Rx ].l ));

            memory->SetByte( memory->GetByte( a[ Rx ].l )+ memory->GetByte( a[ Ry ].l )+ ccr.x,
                             a[ Rx ].l
                           );

            ccc.Rm = BYTE_MSB( memory->GetByte( a[ Rx ].l ));
            ccr.z = memory->GetByte( a[ Rx ].l )== 0;
        }
        break;
    case 1:     // WORD
        if ( !Rm )  // Data Reg. to Data Reg.
        {
            ccc.Sm = WORD_MSB( d[ Ry ].w );
            ccc.Dm = WORD_MSB( d[ Rx ].w );

            d[ Rx ].w += d[ Ry ].w + ccr.x;

            ccc.Rm = WORD_MSB( d[ Rx ].w );
            ccr.z = d[ Rx ].w == 0;
        }
        else      // Address to Address
        {
            a[ Rx ].l -= 2;
            a[ Ry ].l -= 2;
            ccc.Sm = WORD_MSB( memory->GetWord( a[ Ry ].l )); // + ccr.x; ??
            ccc.Dm = WORD_MSB( memory->GetWord( a[ Rx ].l ));

            memory->SetWord( memory->GetWord( a[ Rx ].l )+ memory->GetWord( a[ Ry ].l )+ ccr.x,
                             a[ Rx ].l
                           );

            ccc.Rm = WORD_MSB( memory->GetWord( a[ Rx ].l ));
            ccr.z = memory->GetWord( a[ Rx ].l )== 0;
        }
        break;
    case 2:     // LONGWORD
        if ( !Rm )  // Data Reg. to Data Reg.
        {
            ccc.Sm = LONG_MSB( d[ Ry ].l );
            ccc.Dm = LONG_MSB( d[ Rx ].l );

            d[ Rx ].l += d[ Ry ].l + ccr.x;

            ccc.Rm = LONG_MSB( d[ Rx ].l );
            ccr.z = d[ Rx ].l == 0;
        }
        else      // Address to Address
        {
            a[ Rx ].l -= 4;
```

```
            a[ Ry ].l -= 4;
            ccc.Sm = LONG_MSB( memory->GetLongword( a[ Ry ].l )); // + ccr.x; ??
            ccc.Dm = WORD_MSB( memory->GetLongword( a[ Rx ].l ));

            memory->SetLongword( memory->GetLongword( a[ Rx ].l )+ memory->GetLongword( a[ Ry ].l
                )+ ccr.x,
                        a[ Rx ].l
                       );

            ccc.Rm = LONG_MSB( memory->GetLongword( a[ Rx ].l ));
            ccr.z = memory->GetLongword( a[ Rx ].l )== 0;
          }
        break;
    }
    pc += 2;
}

void M68008::and()
{
    //
    // Alan Donnelly (23/02/2002)
    //

    //
    // This also handles ADDA ( same instruction signature )
    //

    unsigned char Reg = GETBITS(pc, 0xE00, 9);
    unsigned char OpMode = GETBITS(pc, 0x1C0, 6);
    unsigned char EaMode = GETBITS(pc, 0x38, 3);
    unsigned char EaReg = GETBITS(pc, 0x7, 0);
    unsigned long DispedMem = 0;            // Displaced memory address
    DATA_REGISTER( Result );
    DATA_REGISTER( EaValue );
    CCC ccc;

    bool EaIsSource;

    // (d16, An)
    signed short int Disp16;

    // Disp8(An, Xn)
    signed char Disp8;
    unsigned char Xn;
    unsigned char Wl;

    // get addressing mode and source (Ea) value
    EaIsSource = GETBITS(pc, 0x100, 0) == 0;

    switch( OpMode )
    {
    case 0:  // BYTE (Source: Ea)
```

```
ccc.Dm = BYTE_MSB( d[ Reg ].b );

switch( EaMode )
{
case 0:     // Dn.b
   ccc.Sm = BYTE_MSB( d[ EaReg ].b );

   d[ Reg ].b &= d[ EaReg ].b;

   break;
case 1:     // An.b (doesn't apply)
   addx(); return;
   break;
case 2:     // (An).b
case 3:     // (An)+.b
case 4:     // -(An).b
   if( EaMode == 4 )a[ EaReg ].l--;

   EaValue.b = memory->GetByte( a[ EaReg ].l );
   ccc.Sm = BYTE_MSB( EaValue.b );

   d[ Reg ].b &= EaValue.b;

   if( EaMode == 3 )a[ EaReg ].l++;
   break;
case 5:     // (d16, An).b
   Disp16 = memory->GetWord( pc + 2 );
   EaValue.b = memory->GetByte( a[ EaReg ].l + Disp16 );
   ccc.Sm = BYTE_MSB( EaValue.b );

   d[ Reg ].b &= EaValue.b;

   // modify PC
   pc += 2;

   break;
case 6:     // Disp8(An, Xn).b
   //
   // Immediate data for Address Register Indirect with (8-bit Displacement) Mode
   // +--+--+--+-----+---+---+---+---+--+--+--+--+--+--+--+--+--+
   //|   Xn   | w/l  | 0 | 0 | 0 | 0 |       Disp8        |
   // +--+--+--+-----+---+---+---+---+--+--+--+--+--+--+--+--+--+
   //
   // bits between Xn and w/l are scale factor (n/a to add instr.)???
   //

   Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
   Xn = GETBITS( pc + 2, 0x7000, 12 );
   Wl = GETBITS( pc + 2, 0x800, 11 );

   if ( !Wl )      // word index
      EaValue.b = memory->GetByte( a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w );
```

```
    else
        EaValue.b = memory->GetByte( a[ EaReg ].l + Disp8 + d[ Xn ].l );

    ccc.Sm = BYTE_MSB( EaValue.b );
    d[ Reg ].b &= EaValue.b;

    pc += 2;    // modify PC

    break;
case 7:
    switch( EaReg )
    {
        case 0:         // (www).W
            ccc.Sm = BYTE_MSB( memory->GetByte( memory->GetWord( pc + 2 )));
            d[ Reg ].b &= memory->GetByte( memory->GetWord( pc + 2 ));

            pc += 2; // modify PC
            break;
        case 1:         // (www).L
            ccc.Sm = BYTE_MSB( memory->GetByte( memory->GetLongword( pc + 2 )));
            d[ Reg ].b &= memory->GetByte( memory->GetLongword( pc + 2 ));

            pc += 4; // modify PC
            break;
        case 2:         // (d16, PC).b
            Disp16 = memory->GetWord( pc + 2 );
            ccc.Sm = BYTE_MSB( memory->GetByte( pc + Disp16 ));
            d[ Reg ].b &= memory->GetByte( pc + Disp16 );

            pc += 2; // modify PC
            break;
        case 3:         // d8(PC, Xn)
            Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
            Xn = GETBITS( pc + 2, 0x7000, 12 );
            Wl = GETBITS( pc + 2, 0x800, 11 );

            if ( !Wl )    // word index
                EaValue.b = memory->GetByte( pc + Disp8 + (signed short int)d[ Xn ].w );
            else      // long index
                EaValue.b = memory->GetByte( pc + Disp8 + d[ Xn ].l );

            ccc.Sm = BYTE_MSB( EaValue.b );
            d[ Reg ].b &= EaValue.b;

            pc += 2; // modify PC
            break;
        case 4:         // #<data>
            ccc.Sm = BYTE_MSB( memory->GetByte( pc + 3) );
            d[ Reg ].b &= memory->GetByte( pc + 3);

            pc += 2; // modify PC
            break;
```

```
            case 5:
            case 6:
            case 7:
                addx(); return;
                break;
        }
        break;
    }

    // Set carry flag variables
    ccc.Rm = BYTE_MSB( d[ Reg ].b );
    ccr.z = d[ Reg ].b == 0;

    break;

case 1:
    ccc.Dm = WORD_MSB( d[ Reg ].w );

    switch( EaMode )
    {
    // WORD (Source: Ea)
    case 0:     // Dn.w
        ccc.Sm = WORD_MSB( d[ EaReg ].w );
        d[ Reg ].w &= d[ EaReg ].w;
        break;
    case 1:     // An.w
        ccc.Sm = WORD_MSB( a[ EaReg ].w );
        d[ Reg ].w &= a[ EaReg ].w;
        break;
    case 2:     // (An).w
    case 3:     // (An)+.w
    case 4:     // -(An).w
        if( EaMode == 4 )a[ EaReg ].l -= 2;
        EaValue.w = memory->GetWord( a[ EaReg ].l );
        ccc.Sm = WORD_MSB( EaValue.w );

        d[ Reg ].w &= EaValue.w;

        if( EaMode == 3 )a[ EaReg ].l += 2;

        break;
    case 5:  // (d16, An).w
        Disp16 = memory->GetWord( pc + 2 );
        EaValue.w = memory->GetWord( a[ EaReg ].l + Disp16 );
        ccc.Sm = WORD_MSB( EaValue.w );
        d[ Reg ].w &= EaValue.w;

        pc += 2;    // modify PC
        break;
    case 6:     // Disp8(An, Xn).w
        Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
        Xn = GETBITS( pc + 2, 0x7000, 12 );
```

```
    Wl = GETBITS( pc + 2, 0x800, 11 );

    if ( !Wl )        // word index
    {
       EaValue.w = memory->GetWord( a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w );
       ccc.Sm = WORD_MSB( EaValue.w );
       d[ Reg ].w &= EaValue.w;
    }
    else
    {
       EaValue.w = memory->GetWord( a[ EaReg ].l + Disp8 + d[ Xn ].l );
       ccc.Sm = WORD_MSB( EaValue.w );
       d[ Reg ].w &= EaValue.w;
    }

    pc += 2;    // modify PC
    break;
case 7:
    switch( EaReg )
    {
       case 0:          // (www).W
          ccc.Sm = WORD_MSB( memory->GetWord( memory->GetWord( pc + 2 )));
          d[ Reg ].w &= memory->GetWord( memory->GetWord( pc + 2 ));

          pc += 2; // modify PC
          break;
       case 1:          // (www).L
          ccc.Sm = WORD_MSB( memory->GetWord( memory->GetLongword( pc + 2 )));
          d[ Reg ].w &= memory->GetWord( memory->GetLongword( pc + 2 ));

          pc += 4; // modify PC
          break;
       case 2:          // (d16, PC).w
          Disp16 = memory->GetWord( pc + 2 );
          EaValue.w = memory->GetWord( pc + Disp16 );

          ccc.Sm = WORD_MSB( EaValue.w );
          d[ Reg ].w &= EaValue.w;

          pc += 2; // modify PC
          break;
       case 3:          // d8(PC, Xn)
          Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
          Xn = GETBITS( pc + 2, 0x7000, 12 );
          Wl = GETBITS( pc + 2, 0x800, 11 );

          if ( !Wl )    // word index
          {
             EaValue.w = memory->GetWord( pc + Disp8 + (signed short int)d[ Xn ].w );
             ccc.Sm = WORD_MSB( EaValue.w );
             d[ Reg ].w &= EaValue.w;
          }
```

```
            else      // long index
            {
               EaValue.w = memory->GetWord( pc + Disp8 + d[ Xn ].l );
               ccc.Sm = WORD_MSB( EaValue.w );
               d[ Reg ].w &= EaValue.w;
            }

            pc += 2; // modify PC
            break;
         case 4:          // #<data>
            ccc.Sm = WORD_MSB( memory->GetWord( pc + 2) );
            d[ Reg ].w &= memory->GetWord( pc + 2);

            pc += 2; // modify PC
            break;
         case 5:
         case 6:
         case 7:
            addx(); return;
            break;
      }
      break;
   }

   ccc.Rm = WORD_MSB( d[ Reg ].w );
   ccr.z = d[ EaReg ].w == 0;
   break;

case 2:     // LONGWORD (Source: Ea)
   ccc.Dm = LONG_MSB( d[ Reg ].l );

   switch( EaMode )
   {
   case 0:     // Dn.l
      ccc.Sm = LONG_MSB( d[ EaReg ].l );
      d[ Reg ].l &= d[ EaReg ].l;
      break;
   case 1:     // An.l
      ccc.Sm = LONG_MSB( a[ EaReg ].l );
      d[ Reg ].l &= a[ EaReg ].l;
      break;
   case 2:     // (An).l
   case 3:     // (An)+.l
   case 4:     // -(An).l
      if( EaMode == 4 )a[ EaReg ].l -= 4;

      EaValue.l = memory->GetLongword( a[ EaReg ].l );
      ccc.Sm = LONG_MSB( EaValue.l );

      d[ Reg ].l &= EaValue.l;

      if( EaMode == 3 )a[ EaReg ].l += 4;
```

```
    break;
case 5:     // (d16, An).l
    Disp16 = memory->GetWord( pc + 2 );
    EaValue.l = memory->GetLongword( a[ EaReg ].l + Disp16 );
    ccc.Sm = LONG_MSB( EaValue.l );
    d[ Reg ].l &= EaValue.l;

    pc += 2; // modify PC
    break;
case 6:     // Disp8(An, Xn).l
    Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
    Xn = GETBITS( pc + 2, 0x7000, 12 );
    Wl = GETBITS( pc + 2, 0x800, 11 );

    if ( !Wl )   // word index
        EaValue.l = memory->GetLongword( a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w );
    else     // long index
        EaValue.l = memory->GetLongword( a[ EaReg ].l + Disp8 + d[ Xn ].l );

    ccc.Sm = LONG_MSB( EaValue.l );
    d[ Reg ].l &= EaValue.l;

    pc += 2; // modify PC
    break;
case 7:
    switch( EaReg )
    {
        case 0:         // (www).W
            ccc.Sm = LONG_MSB( memory->GetLongword( memory->GetWord( pc + 2 )));
            d[ Reg ].l &= memory->GetLongword( memory->GetWord( pc + 2 ));

            pc += 2; // modify PC
            break;
        case 1:         // (www).L
            ccc.Sm = LONG_MSB( memory->GetLongword( memory->GetLongword( pc + 2 )));
            d[ Reg ].l &= memory->GetLongword( memory->GetLongword( pc + 2 ));

            pc += 4; // modify PC
            break;
        case 2:         // (d16, PC).l
            Disp16 = memory->GetWord( pc + 2 );
            EaValue.l = memory->GetLongword( pc + Disp16 );

            ccc.Sm = LONG_MSB( EaValue.l );
            d[ Reg ].l &= EaValue.l;

            pc += 2; // modify PC
            break;
        case 3:         // d8(PC, Xn)
            Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
            Xn = GETBITS( pc + 2, 0x7000, 12 );
            Wl = GETBITS( pc + 2, 0x800, 11 );
```

```
            if ( !Wl )   // word index
            {
               EaValue.l = memory->GetLongword( pc + Disp8 + (signed short int)d[ Xn ].w );
               ccc.Sm = LONG_MSB( EaValue.l );
               d[ Reg ].l &= EaValue.l;
            }
            else      // long index
            {
               EaValue.l = memory->GetLongword( pc + Disp8 + d[ Xn ].l );
               ccc.Sm = LONG_MSB( EaValue.l );
               d[ Reg ].l &= EaValue.l;
            }

            pc += 2; // modify PC
            break;
         case 4:          // #<data>
            ccc.Sm = LONG_MSB( memory->GetLongword( pc + 2) );
            d[ Reg ].l &= memory->GetLongword( pc + 2);

            pc += 4; // modify PC
            break;
         case 5:
         case 6:
         case 7:
            addx(); return;
            break;
      }
      break;
   }

   ccc.Rm = LONG_MSB( d[ Reg ].l );
   ccr.z = d[ Reg ].l == 0;
   break;

case 4:      // BYTE (Source: Dn)
   ccc.Sm = BYTE_MSB( d[ Reg ].b );

   switch( EaMode )
   {
   case 0:      // Dn.b (n/a for dest = Ea)
   case 1:      // An.b (n/a for dest = Ea)
      addx(); return;
      break;
   case 2:      // (An).b
   case 3:      // (An)+.b
   case 4:      // -(An).b
      if ( EaMode == 4 )a[ EaReg ].l--;

      EaValue.b = memory->GetByte( a[ EaReg ].l );
      ccc.Dm = BYTE_MSB( EaValue.b );
      Result.b = d[ Reg ].b & EaValue.b;
```

```
    memory->SetByte( Result.b, a[ EaReg ].l );

    if( EaMode == 3 )a[ EaReg ].l++;
    break;
case 5:     // (d16, An).b
    Disp16 = memory->GetWord( pc + 2 );

    DispedMem = a[ EaReg ].l + Disp16;
    EaValue.b = memory->GetByte( DispedMem );
    ccc.Dm = BYTE_MSB( EaValue.b );
    Result.b = d[ Reg ].b & EaValue.b;

    memory->SetByte( Result.b, DispedMem );

    pc += 2; // modify PC
    break;
case 6:     // Disp8(An, Xn).b

    Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
    Xn = GETBITS( pc + 2, 0x7000, 12 );
    Wl = GETBITS( pc + 2, 0x800, 11 );

    if ( !Wl )   // word index
       DispedMem = a[ EaReg ].l + (unsigned long)Disp8 + (unsigned long)(signed short int)d[ Xn ].
            w ;
    else     // long index
       DispedMem = a[ EaReg ].l + (unsigned long)Disp8 + d[ Xn ].l;

    EaValue.b = memory->GetByte( DispedMem );

    ccc.Dm = BYTE_MSB( EaValue.b );
    Result.b = d[ Reg ].b & EaValue.b;

    memory->SetByte( Result.b, DispedMem );

    pc += 2; // modify PC
    break;
case 7:
    switch( EaReg )
    {
       case 0:          // (www).W
          ccc.Dm = BYTE_MSB( memory->GetByte( memory->GetWord( pc + 2 )));
          Result.b = d[ Reg ].b & memory->GetByte( memory->GetWord( pc & 2 ));

          memory->SetByte( Result.b, memory->GetWord( pc + 2 ));

          pc += 2; // modify PC
          break;
       case 1:          // (www).L
          ccc.Dm = BYTE_MSB( memory->GetByte( memory->GetLongword( pc + 2 )));
          Result.b = d[ Reg ].b & memory->GetByte( memory->GetLongword( pc + 2 ));
```

```
            memory->SetByte( Result.b, memory->GetLongword( pc + 2 ));

            pc += 4; // modify PC
            break;
         case 2:        // (d16, PC).b
         case 3:        // d8(PC, Xn)
         case 4:        // #<data>
         case 5:
         case 6:
         case 7:
            addx(); return;
            break;
      }
      break;
   }

   ccc.Rm = BYTE_MSB( Result.b );
   ccr.z = Result.b == 0;
   break;

case 5:     // WORD (Source: Dn)
   ccc.Sm = WORD_MSB( d[ Reg ].w );

   switch( EaMode )
   {
   case 0:     // Dn.w (n/a for dest = Ea)
   case 1:     // An.w (n/a for dest = Ea)
      addx(); return;
      break;
   case 2:     // (An).w
   case 3:     // (An)+.w
   case 4:     // -(An).w
      if( EaMode == 4 )a[ EaReg ].l -= 2;

      ccc.Dm = WORD_MSB( memory->GetWord( a[ EaReg ].l ));
      Result.w = d[ Reg ].w & memory->GetWord( a[ EaReg ].l );

      memory->SetWord( Result.w, a[ EaReg ].l );

      if( EaMode == 3 )a[ EaReg ].l += 2;
      break;
   case 5:     // (d16, An).w
      Disp16 = memory->GetWord( pc + 2 );

      DispedMem = a[ EaReg ].l + Disp16;
      EaValue.w = memory->GetWord( DispedMem );

      ccc.Dm = WORD_MSB( EaValue.w );

      Result.w = d[ Reg ].w & EaValue.w;
```

```
    memory->SetWord( Result.w, DispedMem );
    pc += 2; // modify PC
    break;
case 6:     // Disp8(An, Xn).w
    Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
    Xn = GETBITS( pc + 2, 0x7000, 12 );
    Wl = GETBITS( pc + 2, 0x800, 11 );

    if ( !Wl )   // word index
        DispedMem = a[ EaReg ].l + (unsigned long)Disp8 + (unsigned long)(signed short int)d[ Xn ].
            w ;
    else     // long index
        DispedMem = a[ EaReg ].l + (unsigned long)Disp8 + d[ Xn ].l;

    EaValue.w = memory->GetWord( DispedMem );

    ccc.Dm = WORD_MSB( EaValue.w );
    Result.w = d[ Reg ].w & EaValue.w;

    memory->SetByte( Result.b, DispedMem );

    pc += 2; // modify PC
    break;
case 7:
    switch( EaReg )
    {
        case 0:         // (www).W
            ccc.Dm = memory->GetWord( memory->GetWord( pc + 2 ));
            Result.w = d[ Reg ].w & memory->GetWord( memory->GetWord( pc + 2 ));

            memory->SetWord( Result.w, memory->GetWord( pc + 2 ));

            pc += 2; // modify PC
            break;
        case 1:         // (www).L
            ccc.Dm = memory->GetWord( memory->GetLongword( pc + 2 ));
            Result.w = d[ Reg ].w & memory->GetWord( memory->GetLongword( pc + 2 ));

            memory->SetWord( Result.w, memory->GetLongword( pc + 2 ));

            pc += 4; // modify PC
            break;
        case 2:         // (d16, PC).w (n/a for dest = Ea)
        case 3:         // d8(PC, Xn) (n/a for dest = Ea)
        case 4:         // #<data> (n/a for dest = Ea)
        case 5:
        case 6:
        case 7:
            addx(); return;
            break;
    }
    break;
```

```
    }

    ccc.Rm = WORD_MSB( Result.w );
    ccr.z = Result.w == 0;
    break;

case 6:     // LONGWORD (Source: Dn)

    ccc.Sm = LONG_MSB( d[ Reg ].l );

    switch( EaMode )
    {
    case 0:     // Dn.l (n/a for dest = Ea)
    case 1:     // An.l (n/a for dest = Ea)
        addx(); return;
        break;
    case 2:     // (An).l
    case 3:     // (An)+.l
    case 4:     // -(An).l
        if( EaMode == 4 )a[ EaReg ].l -= 4;
        EaValue.l = memory->GetLongword( a[ EaReg ].l );

        ccc.Dm = LONG_MSB( EaValue.l );
        Result.l = d[ Reg ].l & EaValue.l;

        memory->SetLongword( Result.l, a[ EaReg ].l );

        if( EaMode == 3 )a[ EaReg ].l += 4;
        break;
    case 5:     // (d16, An).l
        Disp16 = memory->GetWord( pc + 2 );

        DispedMem = a[ EaReg ].l + Disp16;
        EaValue.l = memory->GetLongword( DispedMem );

        ccc.Dm = LONG_MSB( EaValue.w );
        Result.l = d[ Reg ].l & EaValue.l;

        memory->SetLongword( Result.l, DispedMem );

        pc += 2; // modify PC
        break;
    case 6:     // Disp8(An, Xn).l
        Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
        Xn = GETBITS( pc + 2, 0x7000, 12 );
        Wl = GETBITS( pc + 2, 0x800, 11 );

        if ( !Wl )   // word index
            DispedMem = a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w;
        else     // long index
            DispedMem = a[ EaReg ].l + Disp8 + d[ Xn ].l;
```

```
        EaValue.l = memory->GetLongword( DispedMem );
        ccc.Dm = LONG_MSB( EaValue.l );
        Result.l = d[ Reg ].l & EaValue.l;
        memory->SetLongword( Result.l, DispedMem );

        pc += 2; // modify PC
        break;
    case 7:
        switch( EaReg )
        {
            case 0:         // (www).W
                ccc.Dm = LONG_MSB( memory->GetLongword( memory->GetWord( pc + 2 )));
                Result.l = d[ Reg ].l & memory->GetLongword( memory->GetWord( pc + 2 ));

                memory->SetLongword( Result.l, memory->GetWord( pc + 2 ));

                pc += 2; // modify PC
                break;
            case 1:         // (www).L
                ccc.Dm = LONG_MSB( memory->GetLongword( memory->GetLongword( pc + 2 )));
                Result.l = d[ Reg ].l & memory->GetLongword( memory->GetLongword( pc + 2 ));

                memory->SetLongword( Result.l, memory->GetLongword( pc + 2 ));

                pc += 4; // modify PC
                break;
            case 2:         // (d16, PC).l (n/a for dest = Ea)
            case 3:         // d8(PC, Xn) (n/a for dest = Ea)
            case 4:         // #<data> (n/a for dest = Ea)
            case 5:
            case 6:
            case 7:
                addx(); return;
                break;
        }
        break;
    }
    ccc.Rm = LONG_MSB( Result.l );
    ccr.z = Result.l == 0;
    break;
case 3:
    //
    // Illegal instruction
    //

     illegal ();
    break;
case 7:
    //
    // Illegal instruction
    //
```

```
        illegal ();
        break;
    }


    // Set remaining CCR flags (pp. 89 Motorola Manual)
// ccr.c = ccr.x = ccc.Sm & ccc.Dm | ~ccc.Rm & ccc.Dm | ccc.Sm & ~ccc.Rm;
// ccr.v = ccc.Sm & ccc.Dm & ~ccc.Rm | !ccc.Sm & ~ccc.Dm & ccc.Rm;
// ccr.n = ccc.Rm;

    // Update PC
    pc += 2;
}

void M68008::andi()
{
    //
    // Alan Donnelly and Gerard Whyte (27/02/02)
    //

    unsigned char Size = GET_INSTRBITS(0xC0, 6);
    unsigned char EaMode = GET_INSTRBITS(0x38, 3);
    unsigned char EaReg = GET_INSTRBITS(0x7, 0);

    unsigned long DispedMem = 0;              // Displaced memory address
    DATA_REGISTER( Result );                  // Result of addition
    DATA_REGISTER( Data );                    // Immediate data
    CCC ccc;                                  // Condition code calculations

    // (d16, An)
    signed short int Disp16;

    // Disp8(An, Xn)
    signed char Disp8;
    unsigned char Xn;
    unsigned char Wl;

    switch( Size )
    {
        case 0:      // BYTE
            Data.b = memory->GetWord( pc + 2 )& 0xFF;
            ccc.Sm = BYTE_MSB( Data.b );

            switch( EaMode )
            {
                case 0:      // Dn.b
                    ccc.Dm = BYTE_MSB( d[ EaReg ].b );
                    Result.b = d[ EaReg ].b & Data.b;
                    d[ EaReg ].b = Result.b;

                    break;
                case 1:      // An.b (n/a)
```

```
    illegal ();
    break;
case 2:     // (An).b
case 3:     // (An)++.b
case 4:     // --(An).b
    if( EaMode == 4 )a[ EaReg ].l--;

    ccc.Dm = BYTE_MSB( memory->GetByte( a[ EaReg ].l ));
    Result.b = memory->GetByte( a[ EaReg ].l )& Data.b;
    memory->SetByte( Result.b, a[ EaReg ].l );

    if( EaMode == 3) a[ EaReg ].l++;
    break;
case 5:     // Disp16(An).b
    Disp16 = memory->GetWord( pc + 2 );

    DispedMem = a[ EaReg ].l + Disp16;
    ccc.Dm = BYTE_MSB( memory->GetByte( DispedMem ));
    Result.b = memory->GetByte( DispedMem )& Data.b;
    memory->SetByte( Result.b, DispedMem );

    // Update PC
    pc += 2;
    break;
case 6:     // Disp8(An, Xn).b
    Disp8 = memory->GetByte( pc + 3 );
    Xn = GETBITS( pc + 2, 0x7000, 12 );
    Wl = GETBITS( pc + 2, 0x800, 11 );

    if ( ! Wl )
       DispedMem = a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w;
    else
       DispedMem = a[ EaReg ].l + Disp8 + d[ Xn ].l;

    ccc.Dm = BYTE_MSB( memory->GetByte( DispedMem ));
    Result.b = memory->GetByte( DispedMem )& Data.b;
    memory->SetByte( Result.b, DispedMem );

    // Update PC
    pc += 2;
    break;
case 7:
    switch ( EaReg )
    {
       case 0: // (xxx). W.b
          ccc.Dm = BYTE_MSB( memory->GetByte( memory->GetWord( pc + 2 )));
          Result.b = Data.b & memory->GetByte( memory->GetWord( pc + 2 ));
          memory->SetByte( Result.b, memory->GetWord( pc + 2 ));

          // Update PC
          pc += 2;
          break;
```

```
                case 1: // (xxx).L.b
                    ccc.Dm = BYTE_MSB( memory->GetByte( memory->GetLongword( pc + 2 )));
                    Result.b = Data.b & memory->GetByte( memory->GetLongword( pc + 2 ));
                    memory->SetByte( Result.b, memory->GetLongword( pc + 2 ));

                    // Update PC
                    pc += 4;
                    break;
                default:
                    illegal ();
                    break;
            }
            break;
        }

        ccc.Rm = BYTE_MSB( Result.b );
        ccr.z = Result.b == 0;
        break;

    case 1:     // WORD
        Data.w = memory->GetWord( pc + 2 )& 0xFFFF;
        ccc.Sm = WORD_MSB( Data.w );

        switch( EaMode )
        {
            case 0:     // Dn.w
                ccc.Dm = WORD_MSB( d[ EaReg ].w );
                Result.w = d[ EaReg ].w & Data.w;
                d[ EaReg ].w = Result.w;
                break;
            case 1:     // An.w
                ccc.Dm = WORD_MSB( a[ EaReg ].w );
                Result.w = a[ EaReg ].w & Data.w;
                a[ EaReg ].w += Result.w;
                break;
            case 2:     // (An).w
            case 3:     // (An)++.w
            case 4:     // --(An).w
                if( EaMode == 4 )a[ EaReg ].l -= 2;

                ccc.Dm = WORD_MSB( memory->GetWord( a[ EaReg ].l ));
                Result.w = memory->GetWord( a[ EaReg ].l )& Data.w;
                memory->SetWord( Result.w, a[ EaReg ].l );

                if( EaMode == 3) a[ EaReg ].l += 2;
                break;
            case 5:     // Disp16(An).w
                Disp16 = memory->GetWord( pc + 2 );

                DispedMem = a[ EaReg ].l + Disp16;
                ccc.Dm = WORD_MSB( memory->GetWord( DispedMem ));
                Result.w = memory->GetWord( DispedMem )& Data.w;
```

```
            memory->SetWord( Result.w, DispedMem );

            // Update PC
            pc += 2;
            break;
        case 6:      // Disp8(An, Xn).w
            Disp8 = memory->GetByte( pc + 3 );
            Xn = GETBITS( pc + 2, 0x7000, 12 );
            Wl = GETBITS( pc + 2, 0x800, 11 );

            if ( !Wl )
                DispedMem = a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w;
            else
                DispedMem = a[ EaReg ].l + Disp8 + d[ Xn ].l;

            ccc.Dm = WORD_MSB( memory->GetWord( DispedMem ));
            Result.w = memory->GetWord( DispedMem )& Data.w;

            memory->SetWord( Result.w, DispedMem );

            // Update PC
            pc += 2;
            break;
        case 7:
            switch ( EaReg )
            {
                case 0: // (xxx).W.w
                    ccc.Dm = WORD_MSB( memory->GetWord( memory->GetWord( pc + 2 )));
                    Result.w = Data.w & memory->GetWord( memory->GetWord( pc + 2 ));

                    memory->SetWord( Result.w, memory->GetWord( pc + 2 ));

                    // Update PC
                    pc += 2;
                    break;
                case 1: // (xxx).L.w
                    ccc.Dm = WORD_MSB( memory->GetWord( memory->GetLongword( pc + 2 )));
                    Result.w = Data.w & memory->GetWord( memory->GetLongword( pc + 2 ));

                    memory->SetWord( Result.w, memory->GetLongword( pc + 2 ));

                    // Update PC
                    pc += 4;
                    break;
                default:
                    illegal ();
                    break;
            }
            break;
    }
    ccc.Rm = WORD_MSB( Result.w );
```

```
        ccr.z = Result.w == 0;
        break;

    case 2: // LONGWORD
        Data.l = memory->GetLongword( pc + 2 );

        switch( EaMode )
        {
            case 0:     // Dn.l
                ccc.Dm = LONG_MSB( d[ EaReg ].l );
                Result.l = d[ EaReg ].l & Data.l;
                d[ EaReg ].l = Result.l;
                break;
            case 1:     // An.l
                ccc.Dm = LONG_MSB( a[ EaReg ].l );
                Result.l = a[ EaReg ].l & Data.l;
                a[ EaReg ].l = Result.l;
                break;
            case 2:     // (An).l
            case 3:     // (An)++.l
            case 4:     // --(An).l
                if( EaMode == 4 )a[ EaReg ].l -= 4;

                ccc.Dm = LONG_MSB( memory->GetLongword( a[ EaReg ].l ));
                Result.l = memory->GetLongword( a[ EaReg ].l )& Data.l;
                memory->SetLongword( Result.l, a[ EaReg ].l );

                if( EaMode == 3) a[ EaReg ].l += 4;
                break;
            case 5:     // Disp16(An).l
                Disp16 = memory->GetWord( pc + 2 );
                DispedMem = a[ EaReg ].l + Disp16;

                ccc.Dm = LONG_MSB( memory->GetLongword( DispedMem ));
                Result.l = memory->GetLongword( DispedMem )& Data.l;

                memory->SetLongword( Result.l, DispedMem );

                // Update PC
                pc += 2;
                break;
            case 6:     // Disp8(An, Xn).l
                Disp8 = memory->GetByte( pc + 3 );
                Xn = GETBITS( pc + 2, 0x7000, 12 );
                Wl = GETBITS( pc + 2, 0x800, 11 );

                if  ( !Wl )
                    DispedMem = a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w;
                else
                    DispedMem = a[ EaReg ].l + Disp8 + d[ Xn ].l;

                ccc.Dm = LONG_MSB( memory->GetLongword( DispedMem ));
```

```
                Result.l = memory->GetLongword( DispedMem )& Data.l;
                memory->SetLongword( Result.l, DispedMem );

                // Update PC
                pc += 2;
                break;
            case 7:
                switch ( EaReg )
                {
                    case 0: // (xxx).W.l
                        ccc.Dm = LONG_MSB( memory->GetLongword( memory->GetWord( pc + 2 )));
                        Result.l = Data.l & memory->GetLongword( memory->GetWord( pc + 2 ));
                        memory->SetLongword( Result.l, memory->GetWord( pc + 2 ));

                        // Update PC
                        pc += 2;
                        break;
                    case 1: // (xxx).L.l
                        ccc.Dm = LONG_MSB( memory->GetLongword( memory->GetLongword( pc + 2 )));
                        Result.l = Data.l & memory->GetLongword( memory->GetLongword( pc + 2 ));
                        memory->SetLongword( Result.l, memory->GetLongword( pc + 2 ));

                        // Update PC
                        pc += 4;
                        break;
                    default:
                        illegal ();
                        break;
                }
                break;
        }
        ccc.Rm = LONG_MSB( Result.l );
        ccr.z = Result.l == 0;
        break;
    }

    // Set remaining CCR flags
// ccr.c = ccr.x = ccc.Sm & ccc.Dm | ~ccc.Rm & ccc.Dm | ccc.Sm & ~ccc.Rm;
// ccr.v = ccc.Sm & ccc.Dm & ~ccc.Rm | !ccc.Sm & ~ccc.Dm & ccc.Rm;
// ccr.n = ccc.Rm;

    // Update PC
    pc += 2;
}


void M68008::andiCcr()
{
    unsigned char Data = memory->GetByte( pc + 3 );

    ccr.x = (Data & 0x10) >> 4;
    ccr.n = (Data & 0x8) >> 3;
```

```
    ccr.z = (Data & 0x4) >> 2;
    ccr.v = (Data & 0x2) >> 1;
    ccr.c =  Data & 0x1;

    // Update PC
    pc += 4;

}

void M68008::asl_rReg()
{
    //
    // Gerard Whyte and Alan Donnelly (01/02/02)
    //     Register shifts
    //

    unsigned char Size = GET_INSTRBITS(0xC0, 6);  // Size: b, w, l
    unsigned char Reg = GET_INSTRBITS(0x7, 0);   // Register to shift
    bool left = GET_INSTRBITS( 0x100, 8 )== 1;     // Shift direction
    long int Shift;                                // No of places to shift
    unsigned char vTmp1, vTmp2, Dm;
    unsigned char ImmReg = GET_INSTRBITS(0xE00, 9);  // Either an immediate value for the amount
                                                     // to shift by or the number of the data
                                                     // register holding the shift count

    if ( GET_INSTRBITS( 0x20, 5 ))                   // Register contains shift amount
        Shift = d[ ImmReg ].l;                       // mod 64 according to Motorola manual (pp. 127)
    else
    {
        Shift = ImmReg;                              // Immediate shift value
        if( Shift == 0 ) Shift = 8;
    }

    switch ( Size )
    {
        case 0:          // BYTE
            if( Shift > 0 )
            {
                if ( left ) // Left
                {
                    Dm = BYTE_MSB( d[ Reg ].b );
                    for( int i = 0; i < Shift; i++ )
                    {
                        ccr.c = BYTE_MSB( d[ Reg ].b );
                        d[ Reg ].b <<= 1;

                        vTmp1 |= ~BYTE_MSB( d[ Reg ].b );
                        vTmp2 |= BYTE_MSB( d[ Reg ].b );
                    }

                    ccr.v = Dm & vTmp1 | ~Dm & vTmp2;
                }
```

```
        else      // Right
        {
            d[ Reg ].b >>= ( Shift - 1 );
            ccr.c = d[ Reg ].b & 0x1;
            ccr.v = 0;
            d[ Reg ].b >>= 1;
        }
    }

    ccr.n = BYTE_MSB( d[ Reg ].b );
    ccr.z = d[ Reg ].b == 0;

    break;
case 1:         // WORD
    if( Shift > 0 )
    {
        if ( left ) // Left
        {
            Dm = WORD_MSB( d[ Reg ].w );
            for( int i = 0; i < Shift; i++ )
            {
                ccr.c = WORD_MSB( d[ Reg ].w );
                d[ Reg ].w <<= 1;

                vTmp1 |= ~WORD_MSB( d[ Reg ].w );
                vTmp2 |= WORD_MSB( d[ Reg ].w );
            }

            ccr.v = Dm & vTmp1 | ~Dm & vTmp2;
        }
        else      // Right
        {
            d[ Reg ].w >>= ( Shift - 1 );
            ccr.c = d[ Reg ].w & 0x1;
            ccr.v = 0;
            d[ Reg ].w >>= 1 ;
        }
    }

    ccr.n = WORD_MSB( d[ Reg ].w );
    ccr.z = d[ Reg ].w == 0;
    break;
case 2:         // LONGWORD
    if( Shift > 0 )
    {
        if ( left ) // Left
        {
            Dm = LONG_MSB( d[ Reg ].l );
            for( int i = 0; i < Shift; i++ )
            {
                ccr.c = LONG_MSB( d[ Reg ].l );
                d[ Reg ].l <<= 1;
```

```
                vTmp1 |= ~LONG_MSB( d[ Reg ].l );
                vTmp2 |= LONG_MSB( d[ Reg ].l );
            }

            ccr.v = Dm & vTmp1 | ~Dm & vTmp2;
        }
        else      // Right
        {
            d[ Reg ].l >>= ( Shift - 1 );
            ccr.c = d[ Reg ].l & 0x1;
            ccr.v = 0;
            d[ Reg ].l >>= 1;
        }

    }

    ccr.n = LONG_MSB( d[ Reg ].l );
    ccr.z = d[ Reg ].l == 0;
    break;
  }

  ccr.x = ccr.c;

  // Set remaining flags
  if( Shift == 0 )
    ccr.c = ccr.v = 0;

  // Update the PC
  pc += 2;
}

void M68008::asl_rMem()
{
  //
  // Alan Donnelly and Gerard Whyte (01/02/02)
  // (Note: ASL/R on memory operands only shift once per instruction
  //        and shifts can only be performed on words)
  //

  bool left = GET_INSTRBITS( 0x100, 8 )== 1;  // Direction
  unsigned char EaMode = GET_INSTRBITS(0x38, 3); // Addressing mode
  unsigned char EaReg = GET_INSTRBITS(0x7, 0); // Register
  unsigned short int Operand;                  // Value from memory to be shifted

  // (d16, An)
  signed short int Disp16;

  // Disp8(An, Xn)
  signed char Disp8;
  unsigned char Xn;
  unsigned char Wl;
```

```
switch ( EaMode )
{
   case 0: // Dn (n/a)
   case 1: // An (n/a)
       illegal ();
      break;
   case 2: // (An)
   case 3: // (An)+
   case 4: // -(An)
      if( EaMode == 4 )a[ EaReg ].l -= 2;
      Operand = memory->GetWord( a[ EaReg ].l );

      if ( left )
      {
         ccr.c = Operand & 0x8000;
         memory->SetWord( Operand << 1, a[ EaReg ].l );
      }
      else
      {
         ccr.c = Operand & 0x1;
         memory->SetWord( Operand >> 1, a[ EaReg ].l );
      }

      if( EaMode == 3 )a[ EaReg ].l += 2;
      break;
   case 5: // Disp16(An)
      Disp16 = memory->GetWord( pc + 2);
      Operand = memory->GetWord( a[ EaReg ].l + Disp16 );

      if( left )
      {
         ccr.c = Operand & 0x8000;
         memory->SetWord( Operand << 1, a[ EaReg ].l + Disp16 );
      }
      else
      {
         ccr.c = Operand & 0x1;
         memory->SetWord( Operand >> 1, a[ EaReg ].l + Disp16 );
      }

      // Update PC
      pc += 2;

      break;
   case 6:  // Disp8(An, Xn)
      Disp8 = memory->GetByte( pc + 3 );
      Xn = GETBITS( pc + 2, 0x7000, 12 );
      Wl = GETBITS( pc + 2, 0x800, 11 );

      if ( !Wl )
      {
```

```
            Operand = memory->GetWord( a[ EaReg ].l + Disp8 + d[ Xn ].w );

            if( left )
            {
                ccr.c = Operand & 0x8000;
                memory->SetWord( Operand << 1, a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w );
            }
            else
            {
                ccr.c = Operand & 0x1;
                memory->SetWord( Operand >> 1, a[ EaReg ].l + Disp8 + (signed short int)d[ Xn ].w );
            }
        }
        else
        {
            Operand = memory->GetWord( a[ EaReg ].l + Disp8 + d[ Xn ].l );

            if( left )
            {
                ccr.c = Operand & 0x8000;
                memory->SetWord( Operand << 1, a[ EaReg ].l + Disp8 + d[ Xn ].l );
            }
            else
            {
                ccr.c = Operand &0x1;
                memory->SetWord( Operand >> 1, a[ EaReg ].l + Disp8 + d[ Xn ].l );
            }
        }

        // Update PC
        pc += 2;

        break;

    case 7:
        switch( EaReg )
        {
            case 0:  // (xxx).W
                Operand = memory->GetWord( (unsigned long)memory->GetWord( pc + 2 ));

                if( left )
                {
                    ccr.c = Operand & 0x8000;
                    memory->SetWord( Operand << 1, (unsigned long)memory->GetWord( pc + 2 ));
                }
                else
                {
                    ccr.c = Operand & 0x1;
                    memory->SetWord( Operand >> 1, (unsigned long)memory->GetWord( pc + 2 ));
                }

                // Update PC
```

```
                pc += 2;

                break;
            case 1:  // (xxx).L
                Operand = memory->GetWord( memory->GetLongword( pc + 2 ));

                if( left )
                {
                    ccr.c = Operand & 0x8000;
                    memory->SetWord( Operand << 1, memory->GetLongword( pc + 2 ));
                }
                else
                {
                    ccr.c = Operand & 0x1;
                    memory->SetWord( Operand >> 1, memory->GetLongword( pc + 2 ));
                }

                // Update PC
                pc += 4;

                break;
            default:
                illegal();
                break;
        }
    }

    // Update PC
    pc += 2;
}

void M68008::bxx()
{
    //
    // Alan Donnelly (02/03/02)
    // (Note: This is Bcc in Motorola Manual)
    // Checks the condition and invokes bra()
    //

    //
    // Condition definitions Table 3-19 (pp.90) Motorola Manual
    //
    //  0000 T     0001 F
    //  0010 hi    0011 ls
    //  0100 cc    0101 cs
    //  0110 ne    0111 eq
    //  1000 vc    1001 vs
    //  1010 pl    1011 mi
    //  1100   ge    1101 lt
    //  1110   gt    1111 le
    //
```

```
unsigned char Condition = GET_INSTRBITS(0xF00, 8);

switch( Condition )
{
    case 0:     // True  (not defined)
    case 1:     // False (not defined)
        illegal ();
    case 2:     // BHI (HI = !C && !Z)
        if ( !( ccr.c  ||  ccr.z) )
        {
            bra();
            return;
        }
        break;
    case 3:     // BLS (LS = C || V)
        if ( ccr.c  ||  ccr.z )
        {
            bra();
            return;
        }
        break;
    case 4:     // BCC (CC = !C)
        if ( !ccr.c )
        {
            bra();
            return;
        }
        break;
    case 5:     // BCS (CS = C)
        if ( ccr.c )
        {
            bra();
            return;
        }
        break;
    case 6:     // BNE (NE = !Z)
        if ( !ccr.z )
        {
            bra();
            return;
        }
        break;
    case 7:     // BEQ (EQ = Z)
        if ( ccr.z )
        {
            bra();
            return;
        }
        break;
    case 8:     // BVC (VC = !V)
        if ( !ccr.v )
        {
```

```
            bra();
            return;
        }
        break;
    case 9:       // BVS (VS = V)
        if( ccr.v )
        {
            bra();
            return;
        }
        break;
    case 10: // BPL (PL = !N)
        if( !ccr.n )
        {
            bra();
            return;
        }
        break;
    case 11: // BMI (MI = N)
        if( ccr.n )
        {
            bra();
            return;
        }
        break;
    case 12: // BGE (GE = N && V || !N && !V)
        if( ccr.n && ccr.v || !ccr.n && ccr.v )
        {
            bra();
            return;
        }
        break;
    case 13: // BLT (LT = N && !V || !N && V)
        if( ccr.n && !ccr.v || !ccr.n && ccr.v )
        {
            bra();
            return;
        }
        break;
    case 14: // BGT (GT = N && V && !Z || !N && !V && !Z)
        if( ccr.n && ccr.v && !ccr.z || !ccr.n && !ccr.v && !ccr.z )
        {
            bra();
            return;
        }
        break;
    case 15: // BLE (LE = Z || N && !V || !N && V)
        if( ccr.z || ccr.n && !ccr.v || !ccr.n && ccr.v)
        {
            bra();
            return;
        }
```

```
            break;
    }

    // Nope of the branches was taken, so proceed to the next
    // instruction instead
    char buf[ 32 ];
    pc = Translate( pc, buf );
}

void M68008::btst(unsigned char Bit)
{
    //
    // Alan Donnelly (02/03/02)
    //     auxilary method for btstDyn/Stat
    //     bits tested with bitwise-and and shifting
    //     Z flag set to the bit's value
    //

    unsigned char EaMode = GET_INSTRBITS(0x38, 3);
    unsigned char EaReg = GET_INSTRBITS(0x7, 3);
    unsigned char Operand = 0;
    unsigned long BitMask32 = 0;            // The mask to extract the correct bit (32 bit)
    unsigned char BitMask = 0;              // Same for 8 and 16 bits

    // (d16, An)
    signed short int Disp16;

    // Disp8(An, Xn)
    signed char Disp8;
    unsigned char Xn;
    bool Wl;

    if( EaMode == 0 )
        BitMask32 = 1 << Bit;
    else
        BitMask = 1 << Bit;

    // Bit is used from here onwards to shift the tested bit to the correct position
    // we want to shift this bit Bit - 1 times to the right
    if( Bit > 0 ) Bit--;

    switch( EaMode )
    {
        case 0:     // Dn.l
            ccr.z = ( d[ EaReg ].l & BitMask32 )>> Bit;
            break;
        case 1:     // An (n/a)
            illegal ();
            break;
        case 2:     // (An).b
        case 3:     // (An)+.b
        case 4:     // -(An).b
```

```
    if(EaMode == 4) a[ EaReg ].l--;

    Operand = memory->GetByte( a[ EaReg ].l );
    ccr.z = ( Operand & BitMask )>> Bit;

    if(EaMode == 3) a[ EaReg ].l++;

    break;
case 5:     // (d16, An).b
    Disp16 = memory->GetWord( pc + 2 );
    Operand = memory->GetByte( a[ EaReg ].l + Disp16 );

    ccr.z = ( Operand & BitMask )>> Bit;

    // Update the PC
    pc += 2;

    break;
case 6:     // d8(An, Xn).b
    Disp8 = memory->GetByte( pc + 3 );
    Xn = GETBITS( pc + 2, 0xC0, 4 );
    Wl = memory->GetByte( pc + 2 )& 0x1;

    if( Wl )
    {
        Operand = memory->GetByte( a[ EaReg ].l + Disp8 + d[ Xn ].w );

        ccr.z = ( Operand & BitMask )>> Bit;
    }
    else
    {
        Operand = memory->GetByte( a[ EaReg ].l + Disp8 + d[ Xn ].l );

        ccr.z = ( Operand & BitMask )>> Bit;
    }

    // Update the PC
    pc += 2;
    break;
case 7:
    switch( EaMode )
    {
        case 0:  //(xxx).W.b
            Operand = memory->GetByte( (unsigned long)memory->GetWord( pc + 2 ));

            ccr.z = ( Operand & BitMask )>> Bit;

            // Update the PC
            pc += 2;

            break;
        case 1:  //(xxx).L.b
```

```
                Operand = memory->GetByte( memory->GetLongword( pc + 2 ));

                ccr.z = ( Operand & BitMask )>> Bit;

                // Update the PC
                pc += 4;

                break;
            default:
                illegal ();
                break;
        }

        break;
    }

    // PC updated by calling method
}

void M68008::bset(unsigned char Bit)
{
    //
    // Alan Donnelly (02/03/02)
    //     auxilary method for bsetDyn/Stat
    //     bits set with bitwise-or
    //

    unsigned char EaMode = GET_INSTRBITS(0x38, 3);
    unsigned char EaReg = GET_INSTRBITS(0x7, 3);
    unsigned char Operand = 0;
    unsigned long BitMask32 = 0;
    unsigned char BitMask = 0;

    // (d16, An)
    signed short int Disp16;

    // Disp8(An, Xn)
    signed char Disp8;
    unsigned char Xn;
    bool Wl;

    if( EaMode == 0 )
        BitMask32 = 1 << Bit;
    else
        BitMask = 1 << Bit;

    // Bit is used from here onwards to shift the tested bit to the correct position
    // we want to shift this bit Bit - 1 times to the right
    if( Bit > 0 ) Bit--;

    switch( EaMode )
    {
```

```
case 0:      // Dn.l
   d[ EaReg ].l |= BitMask32;
   break;
case 1:      // An (n/a)
    illegal ();
   break;
case 2:      // (An).b
case 3:      // (An)+.b
case 4:      // -(An).b
   if(EaMode == 4) a[ EaReg ].l--;

   Operand = memory->GetByte( a[ EaReg ].l );
   memory->SetByte(Operand | BitMask, a[ EaReg ].l );

   if(EaMode == 3) a[ EaReg ].l++;

   break;
case 5:      // (d16, An).b
   Disp16 = memory->GetWord( pc + 2 );
   Operand = memory->GetByte( a[ EaReg ].l + Disp16 );
   memory->SetByte( Operand | BitMask, a[ EaReg ].l + Disp16 );

   // Update PC
   pc += 2;

   break;
case 6:      // d8(An, Xn).b
   Disp8 = memory->GetByte( pc + 3 );
   Xn = GETBITS( pc + 2, 0xC0, 4 );
   Wl = memory->GetByte( pc + 2 )& 0x1;

   if( Wl )
   {
      Operand = memory->GetByte( a[ EaReg ].l + Disp8 + d[ Xn ].w );

      memory->SetByte( Operand | BitMask, a[ EaReg ].l + Disp8 + d[ Xn ].w );
   }
   else
   {
      Operand = memory->GetByte( a[ EaReg ].l + Disp8 + d[ Xn ].l );

      memory->SetByte( Operand | BitMask, a[ EaReg ].l + Disp8 + d[ Xn ].l );
   }

   // Update PC
   pc += 2;

   break;
case 7:
   switch( EaMode )
   {
      case 0:  //(xxx).W.b
```

```
                Operand = memory->GetByte( (unsigned long)memory->GetWord( pc + 2 ));

                memory->SetByte( Operand | BitMask, (unsigned long)memory->GetWord( pc + 2 ));

                // Update PC
                pc += 2;

                break;
            case 1:  //(xxx).L.b
                Operand = memory->GetByte( memory->GetLongword( pc + 2 ));

                memory->SetByte( Operand | BitMask, memory->GetLongword( pc + 2 ));

                // Update PC
                pc += 4;

                break;
            default:
                illegal();
                break;
        }

        break;
    }

    // PC updated by calling method
}

void M68008::bclr(unsigned char Bit)
{
    //
    // Alan Donnelly (02/03/02)
    //     auxilary method fro blcrDyn/Stat
    //     bits cleared with bitwise-or
    //

    unsigned char EaMode = GET_INSTRBITS(0x38, 3);
    unsigned char EaReg = GET_INSTRBITS(0x7, 3);
    unsigned char Operand = 0;
    unsigned long BitMask32 = 0;
    unsigned char BitMask = 0;

    // (d16, An)
    signed short int Disp16;

    // Disp8(An, Xn)
    signed char Disp8;
    unsigned char Xn;
    bool Wl;

    if( EaMode == 0 )
        BitMask32 = ~(1 << Bit); // (~ = bitwise NOT)
```

```
else
   BitMask = ~(1 << Bit);

switch( EaMode )
{
   case 0:      // Dn.l
      d[ EaReg ].l &= BitMask32;
      break;
   case 1:      // An (n/a)
       illegal ();
      break;
   case 2:      // (An).b
   case 3:      // (An)+.b
   case 4:      // -(An).b
      if(EaMode == 4) a[ EaReg ].l--;

      Operand = memory->GetByte( a[ EaReg ].l );
      memory->SetByte(Operand & BitMask, a[ EaReg ].l );

      if(EaMode == 3) a[ EaReg ].l++;

      break;
   case 5:      // (d16, An).b
      Disp16 = memory->GetWord( pc + 2 );
      Operand = memory->GetByte( a[ EaReg ].l + Disp16 );
      memory->SetByte( Operand & BitMask, a[ EaReg ].l + Disp16 );

      // Update PC
      pc += 2;
      break;
   case 6:      // d8(An, Xn).b
      Disp8 = memory->GetByte( pc + 3 );
      Xn = GETBITS( pc + 2, 0xC0, 4 );
      Wl = memory->GetByte( pc + 2 )& 0x1;

      if( Wl )
      {
         Operand = memory->GetByte( a[ EaReg ].l + Disp8 + d[ Xn ].w );

         memory->SetByte( Operand & BitMask, a[ EaReg ].l + Disp8 + d[ Xn ].w );
      }
      else
      {
         Operand = memory->GetByte( a[ EaReg ].l + Disp8 + d[ Xn ].l );

         memory->SetByte( Operand & BitMask, a[ EaReg ].l + Disp8 + d[ Xn ].l );
      }

      // Update PC
      pc += 2;

      break;
```

```
        case 7:
            switch( EaMode )
            {
                case 0:  //(xxx).W.b
                    Operand = memory->GetByte( (unsigned long)memory->GetWord( pc + 2 ));

                    memory->SetByte( Operand & BitMask, (unsigned long)memory->GetWord( pc + 2 ));

                    // Update PC
                    pc += 2;

                    break;
                case 1:  //(xxx).L.b
                    Operand = memory->GetByte( memory->GetLongword( pc + 2 ));

                    memory->SetByte( Operand & BitMask, memory->GetLongword( pc + 2 ));

                    // Update PC
                    pc += 4;
                    break;
                default:
                    illegal ();
                    break;
            }

            break;
    }

    // PC updated by calling method

}

void M68008::bchgStat()
{
    //
    // Alan Donnelly (02/03/02)
    // Uses btst, bset and bclr to change the bit
    //

    unsigned char Bit = memory->GetByte( pc + 3 );
    unsigned long SavedPC = 0; // save PC (see bchgDyn())

    SavedPC = pc;
    btst( Bit );
    pc = SavedPC;

    if( ccr.z )
        bset( Bit );
    else
        bclr( Bit );

    // Update PC
```

```
    pc += 4;    // instruction + word static data

}

void M68008::bchgDyn()
{
    //
    // Alan Donnelly (02/03/02)
    // Uses btst, bset and bclr to change the bit
    //

    unsigned char Reg = GET_INSTRBITS(0xE00, 9);
    unsigned char Bit = d[ Reg ].b;
    unsigned long SavedPC = 0;    // need to save PC as btst will modify
                                  // it for (xxx).W etx. addressing modes
                                  // bset and bclr will subsequently modify
                                  // the pc appropriately.
    SavedPC = pc;
    btst( Bit );
    pc = SavedPC;

    if( ccr.z )
        bset( Bit );
    else
        bclr( Bit );

    // update PC
    pc += 2;    // instruction only
}

void M68008::bclrStat()
{
    //
    // Alan Donnelly (02/03/02)
    // uses bclr to clear the bit
    //

    bclr( memory->GetByte( pc + 3 ));

    // update PC
    pc += 4;    // instruction + word static data
}

void M68008::bclrDyn()
{
    //
    // Alan Donnelly (02/03/02)
    // uses bclr to clear the bit

    unsigned char Reg = GET_INSTRBITS(0xE00, 9);

    bclr( d[ Reg ].b );
```

```
    // update PC
    pc += 2;    // instruction only
}


void M68008::bkpt()
{
    //
    // Gerard Whyte
    // For the MC68000 and MC68008, the breakpoint cycle is not run,
    // but an illegal instruction exception is taken.
    //
     illegal ();
}

void M68008::bra()
{
    //
    // Alan Donnelly (02/03/02)
    // (Note: CCR not affected)
    //

    signed char Disp8 = GET_INSTRBITS(0xFF, 0);
    signed short int Disp16 = 0;

    if( Disp8 == -1 )  // 32-bit disp not supported
        illegal ();
    else if ( Disp8 == 0 ) // 16-bit imm. disp if 8-bit disp == 0
    {
        Disp16 = memory->GetWord( pc + 2 );
        pc += Disp16 + 2;
    }
    else                // 8-bit displacement
        pc += Disp8 + 2;
}

void M68008::bsetStat()
{
    //
    // Alan Donnelly (02/03/02)
    // uses bset to set the bit
    //

    bset( memory->GetByte( pc + 3 ));

    // update PC
    pc += 4;    // instruction + word displacement
}

void M68008::bsetDyn()
{
```

```
    //
    // Alan Donnelly (02/03/02)
    // uses bset to set the bit
    //

    unsigned char Reg = GET_INSTRBITS(0xE00, 9);

    bset( d[ Reg ].b );

    // update PC
    pc += 2;    // instruction word only
}

void M68008::bsr()
{
    //
    // Alan Donnelly (02/03/02)
    // (Note: CCR not affected)
    //

    signed char Disp8 = GET_INSTRBITS(0xFF, 0);
    signed short int Disp16 = 0;

    a [ 7 ]. l -= 4;
    PUSH( "Return_Address", 4 );

    pushStack.push( false ); // should this be included ?

    if( Disp8 == -1 )  // 32-bit disp not supported
        illegal ();
    else if ( Disp8 == 0 ) // 16-bit imm. disp if 8-bit disp == 0
    {
        Disp16 = memory->GetWord( pc + 2 );
        memory->SetLongword( pc + 4, a[ 7 ].l );
        pc += Disp16 + 2;
    }
    else                // 8-bit displacement
    {
        memory->SetLongword( pc + 2, a[ 7 ].l );
        pc += Disp8 + 2;
    }

    // Help stack updates
    pushPop.push( a[ 7 ].l );

    // no need to update pc :)
}


void M68008::btstStat()
{
    //
```

```
    // Alan Donnelly (02/03/02)
    // uses btst to test the bit

    btst( memory->GetByte( pc + 3 ));

    // update PC
    pc += 4;    // instruction + word static data
}

void M68008::btstDyn()
{
    //
    // Alan Donnelly (02/03/02)
    // uses btst to test the bit

    unsigned char Reg = GET_INSTRBITS(0xE00, 9);

    bset( d[ Reg ].b );

    // update PC
    pc += 2;    // instruction word only
}

void M68008::chk()
{
    //
    // Gerard Whyte
    // Only need to do WORD operation
    //
    unsigned char Reg = GET_INSTRBITS(0xE00, 9);
    unsigned char EaMode = GET_INSTRBITS(0x38, 3);
    unsigned char EaReg = GET_INSTRBITS(0x7, 0);
    signed short int upper;

    // (d16, An)
    signed short int Disp16;

    // Disp8(An, Xn)
    signed char Disp8;
    unsigned char Xn;
    unsigned char Wl;

    // Does the Stack need to be saved??
    if ( ( (signed short int)d[ Reg ].w < 0 )
    {
        pc = memory->GetLongword( 4 * 6 );
        ccr.n = 1;
    }
    else
    {
        switch ( EaMode )
        {
```

```
case 0: // Dn
    upper = d[ EaReg].w;
    break;
case 1: // An
     illegal ();
    break;
case 2: // (An)
    upper = memory->GetWord( a[ EaReg ].l );
    break;
case 3: // (An)+
    upper = memory->GetWord( a[ EaReg ].l );
    a[ EaReg ].l += 2;
    break;
case 4: // -(An)
    a[ EaReg ].l -= 2;
    upper = memory->GetWord( a[ EaReg ].l );
    break;
case 5: // Disp16(An)
    Disp16 = memory->GetWord( pc + 2 );
    upper = memory->GetWord( a[ EaReg ].l + Disp16 );
    pc += 2;
    break;
case 6: // Disp8(An, Xn)
    Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
    Xn = ( memory->GetByte( pc + 2 )& 0xC0 )>> 4;
    Wl = memory->GetByte( pc + 2 )& 0x1;

    if ( Wl )     // word index
        upper = memory->GetWord( a[ EaReg ].l + Disp8 + d[ Xn ].w );
    else
        upper = memory->GetWord( a[ EaReg ].l + Disp8 + d[ Xn ].l );
    pc += 2;
    break;
case 7:
    switch( EaReg )
    {
        case 0:          // (www).W
            upper = memory->GetWord( memory->GetWord( pc + 2 ));
            pc += 2;
            break;
        case 1:          // (www).L
            upper = memory->GetWord( memory->GetLongword( pc + 2 ));
            pc += 4;
            break;
        case 2:          // (d16, PC).b
            Disp16 = memory->GetWord( pc + 2 );
            upper = memory->GetByte( pc + Disp16 );
            break;
        case 3:          // d8(PC, Xn)
            Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
            Xn = GETBITS( pc + 2, 0xC0, 4 );
            Wl = memory->GetByte( pc + 2 )& 0x1;
```

```
                    if( Wl )     // word index
                        upper = memory->GetByte( pc + Disp8 + d[ Xn ].w );
                    else         // long index
                        upper = memory->GetByte( pc + Disp8 + d[ Xn ].l );
                    break;
                case 4:          // #<data>
                    upper = memory->GetWord( pc + 2);
                    pc += 2;
                    break;
                case 5:
                case 6:
                case 7:
                     illegal ();
                    break;
            }
            break;
        }
        if ( d[ Reg ].w > upper )
        {
            pc = memory->GetLongword( 0x4 * 0x6 );
            ccr.n = 0;
        }
        else
            pc += 2;
    }
}

void M68008::clr()
{
    //
    // Alan Donnelly and Gerard Whyte (25/02/2002)
    //

    //
    // change d[Dn]++ for word and long operations
    //

    unsigned char Size = GET_INSTRBITS(0xC0, 6);
    unsigned char EaMode = GET_INSTRBITS(0x38, 3);
    unsigned char EaReg = GET_INSTRBITS(0x7, 0);

    // (d16, An)
    signed short int Disp16;

    // Disp8(An, Xn)
    signed char Disp8;
    unsigned char Xn;
    unsigned char Wl;

    unsigned long int address;
```

```
// Clear the  register
switch ( ( Size << 3 ) + EaMode )
{
   // BYTE
   case 0:          // Dn.b
      d[EaReg].b = 0;
      break;
   case 1:          // An.b (doesn't apply)
       illegal ();
      break;
   case 2:          // (An).b
      memory->SetByte( 0, a[ EaReg ].l );
      break;
   case 3:          // (An)+.b
      memory->SetByte( 0, a[ EaReg ].l++ );
      break;
   case 4:          // -(An).b
      memory->SetByte( 0, --a[ EaReg ].l );
      break;
   case 5:          // Disp16(An).b
      Disp16 = memory->GetWord( pc + 2 );
      memory->SetByte( 0, a[ EaReg ].l + Disp16 );
      pc += 2;
      break;
   case 6:          // Disp8(An, Xn).b
      Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
      Xn = ( memory->GetByte( pc + 2 )& 0xC0 )>> 4;
      Wl = memory->GetByte( pc + 2 )& 0x1;

      if ( Wl )     // word index
         memory->SetByte( 0, a[ EaReg ].l + Disp8 + d[ Xn ].w );
      else
         memory->SetByte( 0, a[ EaReg ].l + Disp8 + d[ Xn ].l );
      pc += 2;
      break;
   case 7:
      if ( !EaReg )       // (xxx). W.b
      {
         address = memory->GetWord( pc + 2 );
         pc += 2;
      }
      else          // (xxx).L.b
      {
         address = memory->GetLongword( pc + 2 );
         pc += 4;
      }
      memory->SetByte( 0, address );
      break;
   // WORD
   case 8:
      d[EaReg].w = 0;
      break;
```

```
case 9:          // An.w
   a[EaReg].w = 0;
   break;
case 10:         // (An).w
   memory->SetWord( 0, a[ EaReg ].l );
   break;
case 11:         // (An)+.w
   memory->SetWord( 0, a[ EaReg ].l++ );
   break;
case 12:         // -(An).w
   memory->SetWord( 0, --a[ EaReg ].l );
   break;
case 13:         // Disp16(An).w
   Disp16 = memory->GetWord( pc + 2 );
   memory->SetWord( 0, a[ EaReg ].l + Disp16 );
   pc += 2;
   break;
case 14:         // Disp8(An, Xn).w
   Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
   Xn = ( memory->GetByte( pc + 2 )& 0xC0 )>> 4;
   Wl = memory->GetByte( pc + 2 )& 0x1;

   if ( Wl )     // word index
      memory->SetWord( 0, a[ EaReg ].l + Disp8 + d[ Xn ].w );
   else
      memory->SetWord( 0, a[ EaReg ].l + Disp8 + d[ Xn ].l );
   pc += 2;
   break;
case 15:
   if ( !EaReg )       // (xxx).W.w
   {
      address = memory->GetWord( pc + 2 );
      pc += 2;
   }
   else          // (xxx).L.w
   {
      address = memory->GetLongword( pc + 2 );
      pc += 4;
   }
   memory->SetWord( 0, address );
   break;
// LONGWORD
case 16:
   d[EaReg].l = 0;
   break;
case 17:         // An.l
   a[EaReg].l = 0;
   break;
case 18:         // (An).l
   memory->SetLongword( 0, a[ EaReg ].l );
   break;
case 19:         // (An)+.l
```

```
            memory->SetLongword( 0, a[ EaReg ].l++ );
            break;
        case 20:        // -(An).b
            memory->SetLongword( 0, --a[ EaReg ].l );
            break;
        case 21:        // Disp16(An).l
            Disp16 = memory->GetWord( pc + 2 );
            memory->SetLongword( 0, a[ EaReg ].l + Disp16 );
            pc += 2;
            break;
        case 22:        // Disp8(An, Xn).l
            Disp8 = memory->GetByte( pc + 3 ); //skip 1st byte of imm. operand
            Xn = ( memory->GetByte( pc + 2 )& 0xC0 )>> 4;
            Wl = memory->GetByte( pc + 2 )& 0x1;

            if ( Wl )    // word index
                memory->SetLongword( 0, a[ EaReg ].l + Disp8 + d[ Xn ].w );
            else
                memory->SetLongword( 0, a[ EaReg ].l + Disp8 + d[ Xn ].l );
            pc += 2;
            break;
        case 23:
            if ( !EaReg )       // (xxx).W.l
            {
                address = memory->GetWord( pc + 2 );
                pc += 2;
            }
            else            // (xxx).L.l
            {
                address = memory->GetLongword( pc + 2 );
                pc += 4;
            }
            memory->SetLongword( 0, address );
            break;
    }
    // Update the CCR
    ccr.n = 0;
    ccr.v = 0;
    ccr.c = 0;
    ccr.z = 1;

    // Update the PC
    pc += 2;
}

void M68008::cmp()
{
    int mode, opmode, reg, eaReg;
    unsigned long addr;
    DATA_REGISTER( result );

    reg    = GETBITS( pc, 0x0E00, 9 );
```

```
opmode = GETBITS( pc, 0x01C0, 6 );
mode  = GETBITS( pc, 0x0038, 3 );
eaReg = GETBITS( pc, 0x0007, 0 );

ccr.n = ccr.z = ccr.v = ccr.c = 0;

unsigned long int oldPC = pc;

// This instruction gets executed for eor() and cmpa() as well
// Rectifying this here
if ( ( opmode & 3 )== 3 )
{
   cmpa();
   return;
}

if ( opmode >= 4 )
{
   eor();
   return;
}

// dn,dn
if ( mode == 0 )
{
   if ( opmode == 0 )// Byte
   {
      result.b = d[ reg ].b - d[ eaReg ].b;
      SUB_SETOC( d[ eaReg ].b, d[ reg ].b, result.b, MSB8 );
      SUB_SETV( d[ eaReg ].b, d[ reg ].b, result.b, MSB8 );
      SETNZB( result.b );
   }
   if ( opmode == 1 )// Word
   {
      result.w = d[ reg ].w - d[ eaReg ].w;
      SUB_SETOC( d[ eaReg ].w, d[ reg ].w, result.w, MSB16 );
      SUB_SETV( d[ eaReg ].w, d[ reg ].w, result.w, MSB16 );
      SETNZW( result.w );
   }
   if ( opmode == 2 )// Longword
   {
      result.l = d[ reg ].l - d[ eaReg ].l;
      SUB_SETOC( d[ eaReg ].l, d[ reg ].l, result.l, MSB32 );
      SUB_SETV( d[ eaReg ].l, d[ reg ].l, result.l, MSB32 );
      SETNZL( result.l );
   }
   pc += 2;
}

// an,dn
if ( mode == 1 )
{
```

```
    if( opmode == 1 )// Word
    {
        result.w = d[ reg ].w - a[ eaReg ].w;
        SETNZW( result.w );
        SUB_SETOC( a[ eaReg ].w, d[ reg ].w, result.w, MSB16 );
        SUB_SETV( a[ eaReg ].w, d[ reg ].w, result.w, MSB16 );
    }
    if( opmode == 2 )// Longword
    {
        result.l = d[ reg ].l - a[ eaReg ].l;
        SETNZL( result.l );
        SUB_SETOC( a[ eaReg ].l, d[ reg ].l, result.l, MSB32 );
        SUB_SETV( a[ eaReg ].l, d[ reg ].l, result.l, MSB32 );
    }
    pc += 2;
}

// (an),dn
if( mode == 2 )
{
    if( opmode == 0 )// Byte
    {
        result.b = d[ reg ].b - memory->GetByte( a[ eaReg ].l );
        SETNZB( result.b );
        SUB_SETOC( memory->GetByte( a[ eaReg ].l ), d[ reg ].b, result.b, MSB8 );
        SUB_SETV( memory->GetByte( a[ eaReg ].l ), d[ reg ].b, result.b, MSB8 );
    }
    if( opmode == 1 )// Word
    {
        result.w = d[ reg ].w - memory->GetWord( a[ eaReg ].l );
        SETNZW( result.w );
        SUB_SETOC( memory->GetWord( a[ eaReg ].l ), d[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( a[ eaReg ].l ), d[ reg ].w, result.w, MSB16 );
    }
    if( opmode == 2 )// Longword
    {
        result.l = d[ reg ].l - memory->GetLongword( a[ eaReg ].l );
        SETNZL( result.l );
        SUB_SETOC( memory->GetLongword( a[ eaReg ].l ), d[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( a[ eaReg ].l ), d[ reg ].l, result.l, MSB32 );
    }
    pc += 2;
}

// (an)+,dn
if( mode == 3 )
{
    if( opmode == 0 )// Byte
    {
        result.b = d[ reg ].b - memory->GetByte( a[ eaReg ].l );
        SETNZB( result.b );
        SUB_SETOC( memory->GetByte( a[ eaReg ].l ), d[ reg ].b, result.b, MSB8 );
```

```
        SUB_SETV( memory->GetByte( a[ eaReg ].l ), d[ reg ].b, result.b, MSB8 );
        a[ eaReg ].l += 1;
    }
    if( opmode == 1 )// Word
    {
        result.w = d[ reg ].w - memory->GetWord( a[ eaReg ].l );
        SETNZW( result.w );
        SUB_SETOC( memory->GetWord( a[ eaReg ].l ), d[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( a[ eaReg ].l ), d[ reg ].w, result.w, MSB16 );
        a[ eaReg ].l += 2;
    }
    if( opmode == 2 )// Longword
    {
        result.l = d[ reg ].l - memory->GetLongword( a[ eaReg ].l );
        SETNZL( result.l );
        SUB_SETOC( memory->GetLongword( a[ eaReg ].l ), d[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( a[ eaReg ].l ), d[ reg ].l, result.l, MSB32 );
        a[ eaReg ].l += 4;
    }
    pc += 2;
}

// -(an),dn
if( mode == 4 )
{
    if( opmode == 0 )// Byte
    {
        a[ eaReg ].l -= 1;
        result.b = d[ reg ].b - memory->GetByte( a[ eaReg ].l );
        SETNZB( result.b );
        SUB_SETOC( memory->GetByte( a[ eaReg ].l ), d[ reg ].b, result.b, MSB8 );
        SUB_SETV( memory->GetByte( a[ eaReg ].l ), d[ reg ].b, result.b, MSB8 );
    }
    if( opmode == 1 )// Word
    {
        a[ eaReg ].l -= 2;
        result.w = d[ reg ].w - memory->GetWord( a[ eaReg ].l );
        SETNZW( result.w );
        SUB_SETOC( memory->GetWord( a[ eaReg ].l ), d[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( a[ eaReg ].l ), d[ reg ].w, result.w, MSB16 );
    }
    if( opmode == 2 )// Longword
    {
        a[ eaReg ].l -= 4;
        result.l = d[ reg ].l - memory->GetLongword( a[ eaReg ].l );
        SETNZL( result.l );
        SUB_SETOC( memory->GetLongword( a[ eaReg ].l ), d[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( a[ eaReg ].l ), d[ reg ].l, result.l, MSB32 );
    }
    pc += 2;

}
```

```
// d16(an),dn
if( mode == 5 )
{
    unsigned long addr = a[ eaReg ].l + (signed short int)memory->GetWord( pc + 2 );
    if( opmode == 0 )// Byte
    {
        result.b = d[ reg ].b - memory->GetByte( addr );
        SETNZB( result.b );
        SUB_SETOC( memory->GetByte( addr ), d[ reg ].b, result.b, MSB8 );
        SUB_SETV( memory->GetByte( addr ), d[ reg ].b, result.b, MSB8 );
    }
    if( opmode == 1 )// Word
    {
        result.w = d[ reg ].w - memory->GetWord( addr );
        SETNZW( result.w );
        SUB_SETOC( memory->GetWord( addr ), d[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( addr ), d[ reg ].w, result.w, MSB16 );
    }
    if( opmode == 2 )// Longword
    {
        result.l = d[ reg ].l - memory->GetLongword( addr );
        SETNZL( result.l );
        SUB_SETOC( memory->GetLongword( addr ), d[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( addr ), d[ reg ].l, result.l, MSB32 );
    }
    pc += 4;
}

// d8(an,Xn),dn
if( mode == 6 )
{
    addr = a[ eaReg ].l + (signed char)memory->GetByte( pc + 3 )
        + d[ GETBITS( pc + 2, 0xF000, 12 )].l;

    if( opmode == 0 )// Byte
    {
        result.b = d[ reg ].b - memory->GetByte( addr );
        SETNZB( result.b );
        SUB_SETOC( memory->GetByte( addr ), d[ reg ].b, result.b, MSB8 );
        SUB_SETV( memory->GetByte( addr ), d[ reg ].b, result.b, MSB8 );
    }
    if( opmode == 1 )// Word
    {
        result.w = d[ reg ].w - memory->GetWord( addr );
        SETNZW( result.w );
        SUB_SETOC( memory->GetWord( addr ), d[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( addr ), d[ reg ].w, result.w, MSB16 );
    }
    if( opmode == 2 )// Longword
    {
        result.l = d[ reg ].l - memory->GetLongword( addr );
```

```
            SETNZL( result.l );
            SUB_SETOC( memory->GetLongword( addr ), d[ reg ].l, result.l, MSB32 );
            SUB_SETV( memory->GetLongword( addr ), d[ reg ].l, result.l, MSB32 );
        }
        pc += 4;
    }

    //(XXX).W,dn
    if ( mode == 7 && eaReg < 2 )
    {
        if ( eaReg == 0 )
        {
            addr = memory->GetWord( pc + 2 );
            pc += 4;
        }
        else
        {
            addr = memory->GetLongword( pc + 2 );
            pc += 6;
        }
        if ( opmode == 0 )// Byte
        {
            result.b = d[ reg ].b - memory->GetByte( addr );
            SETNZB( result.b );
            SUB_SETOC( memory->GetByte( addr ), d[ reg ].b, result.b, MSB8 );
            SUB_SETV( memory->GetByte( addr ), d[ reg ].b, result.b, MSB8 );
        }
        if ( opmode == 1 )// Word
        {
            result.w = d[ reg ].w - memory->GetWord( addr );
            SETNZW( result.w );
            SUB_SETOC( memory->GetWord( addr ), d[ reg ].w, result.w, MSB16 );
            SUB_SETV( memory->GetWord( addr ), d[ reg ].w, result.w, MSB16 );
        }
        if ( opmode == 2 )// Longword
        {
            result.l = d[ reg ].l - memory->GetLongword( addr );
            SETNZL( result.l );
            SUB_SETOC( memory->GetLongword( addr ), d[ reg ].l, result.l, MSB32 );
            SUB_SETV( memory->GetLongword( addr ), d[ reg ].l, result.l, MSB32 );
        }
    }

    // #<data>,dn
    if ( mode == 7 && eaReg == 4)
    {
        if ( opmode == 0 )// Byte
        {
            result.b = d[ reg ].b - memory->GetByte( pc + 3 );
            SETNZB( result.b );
            SUB_SETOC( memory->GetByte( pc + 3 ), d[ reg ].b, result.b, MSB8 );
            SUB_SETV( memory->GetByte( pc + 3 ), d[ reg ].b, result.b, MSB8 );
```

```
            pc += 2;
        }
        if( opmode == 1 )// Word
        {
            result.w = d[ reg ].w - memory->GetWord( pc + 2 );
            SETNZW( result.w );
            SUB_SETOC( memory->GetWord( pc + 2 ), d[ reg ].w, result.w, MSB16 );
            SUB_SETV( memory->GetWord( pc + 2 ), d[ reg ].w, result.w, MSB16 );
            pc += 2;
        }
        if( opmode == 2 )// Longword
        {
            result.l = d[ reg ].l - memory->GetLongword( pc + 2 );
            SETNZL( result.l );
            SUB_SETOC( memory->GetLongword( pc + 2 ), d[ reg ].l, result.l, MSB32 );
            SUB_SETV( memory->GetLongword( pc + 2 ), d[ reg ].l, result.l, MSB32 );
            pc += 4;
        }
        pc += 2;
    }

    // OMITTED
    // ( D16, PC )
    // ( d8, PC, Xn )

    // The following code should be put after the execution of the CMP instruction
    // It needs the PC to be updated to point to the instruction after the CMP instruction
    char *buf;
    if( GETBITS( pc, 0xF000, 12 )!= 6 )  // not followed by a Bcc?
    {
        buf = new char[ 128 ];
        sprintf( buf, "Warning at address $%lX: You probably want to follow\n a CMP with a Bcc instruction
            (e.g. BEQ)", oldPC );
        helpStack.push( buf );
    }
}

void M68008::cmpa()
{
    int mode, opmode, reg, eaReg;
    unsigned long addr;
    ADDRESS_REGISTER( result );
    ADDRESS_REGISTER( imm );

    reg = GETBITS( pc, 0x0E00, 9 );
    opmode = GETBITS( pc, 0x01C0, 6 );
    mode = GETBITS( pc, 0x0038, 3 );
    eaReg = GETBITS( pc, 0x0007, 0 );

    ccr.n = ccr.z = ccr.v = ccr.c = 0;

    unsigned long int oldPC = pc;
```

```
// dn,an
if( mode == 0 )
{
    if( opmode == 3 )// Word
    {
        result.w = a[ reg ].w - d[ eaReg ].w;
        SUB_SETOC( d[ eaReg ].w, a[ reg ].w, result.w, MSB16 );
        SUB_SETV( d[ eaReg ].w, a[ reg ].w, result.w, MSB16 );
        SETNZW( result.w );
    }
    if( opmode == 7 )// Longword
    {
        result.l = a[ reg ].l - d[ eaReg ].l;
        SUB_SETOC( d[ eaReg ].l, a[ reg ].l, result.l, MSB32 );
        SUB_SETV( d[ eaReg ].l, a[ reg ].l, result.l, MSB32 );
        SETNZL( result.l );
    }
    pc += 2;
}
// an, an
if( mode == 1 )
{
    if( opmode == 3 )// Word
    {
        result.w = a[ reg ].w - a[ eaReg ].w;
        SUB_SETOC( a[ reg ].w, d[ eaReg ].w, result.w, MSB16 );
        SUB_SETV( a[ reg ].w, d[ eaReg ].w, result.w, MSB16 );
        SETNZW( result.w );
    }
    if( opmode == 7 )// Longword
    {
        result.l = a[ reg ].l - a[ eaReg ].l;
        SUB_SETOC( a[ reg ].l, d[ eaReg ].l, result.l, MSB32 );
        SUB_SETV( a[ reg ].l, d[ eaReg ].l, result.l, MSB32 );
        SETNZL( result.l );
    }
    pc += 2;
}
// ( an ), an
if( mode == 2 )
{
    if( opmode == 3 )// Word
    {
        result.w = (unsigned)a[ reg ].w - memory->GetWord( a[ eaReg ].l );
        SUB_SETOC( memory->GetWord( a[ eaReg ].l ), (unsigned)a[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( a[ eaReg ].l ), (unsigned)a[ reg ].w, result.w, MSB16 );
        SETNZW( result.w );
    }
    if( opmode == 7 )// Longword
    {
        result.l = a[ reg ].l - memory->GetLongword( a[ eaReg ].l);
```

```
        SUB_SETOC( memory->GetLongword( a[ eaReg ].l ), (unsigned)a[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( a[ eaReg ].l ), (unsigned)a[ reg ].l, result.l, MSB32 );
        SETNZL( result.l );
    }
    pc += 2;
}
// ( an )+, an
if( mode == 3 )
{
    if( opmode == 3 )// Word
    {
        result.w = (unsigned)a[ reg ].w - memory->GetWord( a[ eaReg ].l );
        SUB_SETOC( memory->GetWord( a[ eaReg ].l ), (unsigned)a[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( a[ eaReg ].l ), (unsigned)a[ reg ].w, result.w, MSB16 );
        SETNZW( result.w );
        a[ eaReg ].l += 2;

    }
    if( opmode == 7 )// Longword
    {
        result.l = a[ reg ].l - memory->GetLongword( a[ eaReg ].l);
        SUB_SETOC( memory->GetLongword( a[ eaReg ].l ), (unsigned)a[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( a[ eaReg ].l ), (unsigned)a[ reg ].l, result.l, MSB32 );
        SETNZL( result.l );
        a[ eaReg ].l += 4;
    }
    pc += 2;
}
// -( an ), an
if( mode == 4 )
{
    if( opmode == 3 )// Word
    {
        a[ eaReg ].l -= 2;
        result.w = (unsigned)a[ reg ].w - memory->GetWord( a[ eaReg ].l );
        SUB_SETOC( memory->GetWord( a[ eaReg ].l ), (unsigned)a[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( a[ eaReg ].l ), (unsigned)a[ reg ].w, result.w, MSB16 );
        SETNZW( result.w );

    }
    if( opmode == 7 )// Longword
    {
        a[ eaReg ].l -= 4;
        result.l = a[ reg ].l - memory->GetLongword( a[ eaReg ].l);
        SUB_SETOC( memory->GetLongword( a[ eaReg ].l ), (unsigned)a[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( a[ eaReg ].l ), (unsigned)a[ reg ].l, result.l, MSB32 );
        SETNZL( result.l );

    }
    pc += 2;
}
// d16(an), an
```

```
if ( mode == 5 )
{
    addr = a[ eaReg ].l + (signed short int)memory->GetWord( pc + 2 );
    if ( opmode == 3 )// Word
    {
        result.w = a[ reg ].w - memory->GetWord( addr );
        SUB_SETOC( memory->GetWord( addr ), a[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( addr ), a[ reg ].w, result.w, MSB16 );
        SETNZW( result.w );
    }
    if ( opmode == 7 )// Longword
    {
        result.l = a[ reg ].l - memory->GetLongword( addr );
        SUB_SETOC( memory->GetLongword( addr ), a[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( addr ), a[ reg ].l, result.l, MSB32 );
        SETNZL( result.l );
    }
    pc += 4;
}
// d8(an), an
if ( mode == 6 )
{
    addr = a[ eaReg ].l + (signed char)memory->GetByte( pc + 3 )
        + d[ GETBITS( pc + 2, 0xf000, 12 )].l;
    if ( opmode == 3 )// Word
    {
        result.w = a[ reg ].w - memory->GetWord( addr );
        SUB_SETOC( memory->GetWord( addr ), a[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( addr ), a[ reg ].w, result.w, MSB16 );
        SETNZW( result.w );
    }
    if ( opmode == 7 )// Longword
    {
        result.l = a[ reg ].l - memory->GetLongword( addr );
        SUB_SETOC( memory->GetLongword( addr ), a[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( addr ), a[ reg ].l, result.l, MSB32 );
        SETNZL( result.l );
    }
    pc += 4;
}

// #<data>,an
if ( ( mode == 7 )&& ( eaReg == 4 ))
{
    if ( opmode == 3 )// Word
    {
        imm.w = (signed short int)memory->GetWord( pc + 2 );
        result.w = a[ reg ].w - imm.w;
        SETNZW( result.w );
        SUB_SETOC( imm.w, a[ reg ].w, result.w, MSB16 );
        SUB_SETV( imm.w, a[ reg ].w, result.w, MSB16 );
        pc += 4;
```

```
    }
    if ( opmode == 7 )// Longword
    {
        imm.l = (signed long int)memory->GetLongword( pc + 2 );
        result.l = a[ reg ].l - imm.l;
        SETNZL( result.l );
        SUB_SETOC( imm.l, a[ reg ].l, result.l, MSB32 );
        SUB_SETV( imm.l, a[ reg ].l, result.l, MSB32 );
        pc += 6;
    }
    return;
}

//(XXX).W, an
if ( mode == 7 )
{
    if ( eaReg == 0 )
    {
        addr = memory->GetWord( pc + 2 );
        pc += 4;
    }
    else
    {
        //(XXX).L, an
        addr = memory->GetLongword( pc + 2 );
        pc += 6;
    }
    if ( opmode == 3 )// Word
    {
        result.w = a[ reg ].w
            - memory->GetWord( addr );
        SETNZW( result.w );
        SUB_SETOC( memory->GetWord( addr ), a[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( addr ), a[ reg ].w, result.w, MSB16 );
    }
    if ( opmode == 7 )// Longword
    {
        result.l = a[ reg ].l
            - memory->GetLongword( addr );
        SETNZL( result.l );
        SUB_SETOC( memory->GetLongword( addr ), a[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( addr ), a[ reg ].l, result.l, MSB32 );
    }
}

// OMITTED
// ( D16, PC )
// ( d8, PC, Xn )

// The following code should be put after the execution of the CMP instruction
// It needs the PC to be updated to point to the instruction after the CMP instruction
char *buf;
```

```
    if( GETBITS( pc, 0xF000, 12 )!= 6 )  // not followed by a Bcc?
    {
        buf = new char[ 128 ];
        sprintf( buf, "WARNING at address $%lX: You probably want to follow\n a CMP with a Bcc
            instruction (e.g. BEQ)", oldPC );
        helpStack.push( buf );
    }
}

void M68008::cmpi()
{
    int size, mode, reg, displ;
    DATA_REGISTER( result );
    DATA_REGISTER( imm );

    size = GETBITS( pc, 0x00C0, 6 );
    mode = GETBITS( pc, 0x0038, 3 );
    reg = GETBITS( pc, 0x0007, 0 );

    ccr.n = ccr.z = ccr.v = ccr.c = 0;

    unsigned long int oldPC = pc;

    // #data,dn
    if( mode == 0 )
    {
        if( size == 0 ) // Byte
        {
            imm.b = memory->GetByte( pc + 3 );
            result.b = d[ reg ].b - imm.b;
            SETNZB( result.b );
            SUB_SETOC( imm.b, d[ reg ].b, result.b, MSB8 );
            SUB_SETV( imm.b, d[ reg ].b, result.b, MSB8 );
            pc += 4;
        }
        if( size == 1 ) // Word
        {
            imm.w = memory->GetWord( pc + 2 );
            result.w = d[ reg ].w - imm.w;
            SETNZW( result.w );
            SUB_SETOC( imm.w, d[ reg ].w, result.w, MSB16 );
            SUB_SETV( imm.w, d[ reg ].w, result.w, MSB16 );
            pc += 4;
        }
        if( size == 2 ) // Longword
        {
            imm.l = memory->GetLongword( pc + 2 );
            result.l = d[ reg ].l - imm.l;
            SETNZL( result.l );
            SUB_SETOC( imm.l, d[ reg ].l, result.l, MSB32 );
            SUB_SETV( imm.l, d[ reg ].l, result.l, MSB32 );
            pc += 6;
```

```
    }
}

// #data,(an)
if( mode == 2 )
{
    if( size == 0 ) // Byte
    {
        imm.b = memory->GetByte( pc + 3 );
        result.b = memory->GetByte( a[ reg ].l )- imm.b;
        SETNZB( result.b );
        SUB_SETOC( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        SUB_SETV( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        pc += 4;
    }
    if( size == 1 ) // Word
    {
        imm.w = memory->GetWord( pc + 2 );
        result.w = memory->GetWord( a[ reg ].l )- imm.w;
        SETNZW( result.w );
        SUB_SETOC( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        SUB_SETV( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        pc += 4;
    }
    if( size == 2 ) // Longword
    {
        imm.l = memory->GetLongword( pc + 2 );
        result.l = memory->GetLongword( a[ reg ].l )- imm.l;
        SETNZL( result.l );
        SUB_SETOC( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        SUB_SETV( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        pc += 6;
    }
}

// #data, (an)+
if( mode == 3 )
{
    if( size == 0 ) // Byte
    {
        imm.b = memory->GetByte( pc + 3 );
        result.b = memory->GetByte( a[ reg ].l )- imm.b;
        SETNZB( result.b );
        SUB_SETOC( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        SUB_SETV( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        pc += 4;
        a[ reg ].l += 1;
    }
    if( size == 1 ) // Word
    {
        imm.w = memory->GetWord( pc + 2 );
        result.w = memory->GetWord( a[ reg ].l )- imm.w;
```

```
        SETNZW( result.w );
        SUB_SETOC( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        SUB_SETV( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        pc += 4;
        a[ reg ].l += 2;
    }
    if( size == 2 ) // Longword
    {
        imm.l = memory->GetLongword( pc + 2 );
        result.l = memory->GetLongword( a[ reg ].l )- imm.l;
        SETNZL( result.l );
        SUB_SETOC( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        SUB_SETV( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        pc += 6;
        a[ reg ].l += 4;
    }
}

// #data, -(an)
if( mode == 4 )
{
    if( size == 0 ) // Byte
    {
        a[ reg ].l -= 1;
        imm.b = memory->GetByte( pc + 3 );
        result.b = memory->GetByte( a[ reg ].l )- imm.b;
        SETNZB( result.b );
        SUB_SETOC( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        SUB_SETV( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        pc += 4;
    }
    if( size == 1 ) // Word
    {
        a[ reg ].l -= 2;
        imm.w = memory->GetWord( pc + 2 );
        result.w = memory->GetWord( a[ reg ].l )- imm.w;
        SETNZW( result.w );
        SUB_SETOC( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        SUB_SETV( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        pc += 4;
    }
    if( size == 2 ) // Longword
    {
        a[ reg ].l -= 4;
        imm.l = memory->GetLongword( pc + 2 );
        result.l = memory->GetLongword( a[ reg ].l )- imm.l;
        SETNZL( result.l );
        SUB_SETOC( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        SUB_SETV( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        pc += 6;
    }
}
```

```
// #data, d16(an)
if ( mode == 5 )
{
    if ( size == 0 ) // Byte
    {
        displ = (signed short int)memory->GetWord( pc + 4 );
        imm.b = memory->GetByte( pc + 3 );
        result.b = memory->GetByte( a[ reg ].l + displ )- imm.b;
        SETNZB( result.b );
        SUB_SETOC( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        SUB_SETV( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        pc += 6;
    }
    if ( size == 1 ) // Word
    {
        displ = (signed short int)memory->GetWord( pc + 4 );
        imm.w = memory->GetWord( pc + 2 );
        result.w = memory->GetWord( a[ reg ].l + displ )- imm.w;
        SETNZW( result.w );
        SUB_SETOC( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        SUB_SETV( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        pc += 6;
    }
    if ( size == 2 ) // Longword
    {
        displ = (signed short int)memory->GetWord( pc + 6 );
        imm.l = memory->GetLongword( pc + 2 );
        result.l = memory->GetLongword( a[ reg ].l + displ )- imm.l;
        SETNZL( result.l );
        SUB_SETOC( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        SUB_SETV( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        pc += 8;
    }
}

// #data, d8(an,xn)
if ( mode == 6 )
{
    if ( size == 0 ) // Byte
    {
        displ = (signed char)memory->GetByte( pc + 5 )
            + d[ GETBITS( pc + 4, 0xF000, 12 )].l;
        imm.b = memory->GetByte( pc + 3 );
        result.b = memory->GetByte( a[ reg ].l + displ )- imm.b;
        SETNZB( result.b );
        SUB_SETOC( imm.b, memory->GetByte( a[ reg ].l + displ ), result.b, MSB8 );
        SUB_SETV( imm.b, memory->GetByte( a[ reg ].l + displ ), result.b, MSB8 );
        pc += 6;
    }
    if ( size == 1 ) // Word
    {
```

```
            displ = (signed char)memory->GetByte( pc + 5 )
                + d[ GETBITS( pc + 4, 0xF000, 12 )].l;
            imm.w = memory->GetWord( pc + 2 );
            result.w = memory->GetWord( a[ reg ].l + displ ) - imm.w;
            SETNZW( result.w );
            SUB_SETOC( imm.w, memory->GetWord( a[ reg ].l + displ ), result.w, MSB16 );
            SUB_SETV( imm.w, memory->GetWord( a[ reg ].l + displ ), result.w, MSB16 );
            pc += 6;
        }
        if( size == 2 ) // Longword
        {
            displ = (signed char)memory->GetByte( pc + 7 )
                + d[ GETBITS( pc + 4, 0xF000, 12 )].l;
            imm.l = memory->GetLongword( pc + 2 );
            result.l = memory->GetLongword( a[ reg ].l + displ ) - imm.l;
            SETNZL( result.l );
            SUB_SETOC( imm.l, memory->GetLongword( a[ reg ].l + displ ), result.l, MSB32 );
            SUB_SETV( imm.l, memory->GetLongword( a[ reg ].l + displ ), result.l, MSB32 );
            pc += 8;
        }
    }

    if( mode == 7 )
    {
        // #data, (xxx).w
        if( reg == 0 )
        {
            if( size == 0 ) // Byte
            {
                displ = memory->GetWord( pc + 4 );
                imm.b = memory->GetByte( pc + 3 );
                result.b = memory->GetByte( displ ) - imm.b;
                SETNZB( result.b );
                SUB_SETOC( imm.b, memory->GetByte( displ ), result.b, MSB8 );
                SUB_SETV( imm.b, memory->GetByte( displ ), result.b, MSB8 );
                pc += 6;
            }
            if( size == 1 ) // Word
            {
                displ = memory->GetWord( pc + 4 );
                imm.w = memory->GetWord( pc + 2 );
                result.w = memory->GetWord( displ ) - imm.w;
                SETNZW( result.w );
                SUB_SETOC( imm.w, memory->GetWord( displ ), result.w, MSB16 );
                SUB_SETV( imm.w, memory->GetWord( displ ), result.w, MSB16 );
                pc += 6;
            }
            if( size == 2 ) // Longword
            {
                displ = memory->GetWord( pc + 6 );
                imm.l = memory->GetLongword( pc + 2 );
                result.l = memory->GetLongword( displ ) - imm.l;
```

```
            SETNZL( result.l );
            SUB_SETOC( imm.l, memory->GetLongword( displ ), result.l, MSB32 );
            SUB_SETV( imm.l, memory->GetLongword( displ ), result.l, MSB32 );
            pc += 8;
        }
    }

    // #data, (xxx).l
    if( reg == 1 )
    {
        if( size == 0 ) // Byte
        {
            displ = memory->GetLongword( pc + 4 );
            imm.b = memory->GetByte( pc + 3 );
            result.b = memory->GetByte( displ )- imm.b;
            SETNZB( result.b );
            SUB_SETOC( imm.b, memory->GetByte( displ ), result.b, MSB8 );
            SUB_SETV( imm.b, memory->GetByte( displ ), result.b, MSB8 );
            pc += 8;
        }
        if( size == 1 ) // Word
        {
            displ = memory->GetLongword( pc + 4 );
            imm.w = memory->GetWord( pc + 2 );
            result.w = memory->GetWord( displ )- imm.w;
            SETNZW( result.w );
            SUB_SETOC( imm.w, memory->GetWord( displ ), result.w, MSB16 );
            SUB_SETV( imm.w, memory->GetWord( displ ), result.w, MSB16 );
            pc += 8;
        }
        if( size == 2 ) // Longword
        {
            displ = memory->GetLongword( pc + 6 );
            imm.l = memory->GetLongword( pc + 2 );
            result.l = memory->GetLongword( displ )- imm.l;
            SETNZL( result.l );
            SUB_SETOC( imm.l, memory->GetLongword( displ ), result.l, MSB32 );
            SUB_SETV( imm.l, memory->GetLongword( displ ), result.l, MSB32 );
            pc += 10;
        }
    }
}

// OMITTED
// ( D16, PC )
// ( d8, PC, Xn )

// The following code should be put after the execution of the CMP instruction
// It needs the PC to be updated to point to the instruction after the CMP instruction
char *buf;
if( GETBITS( pc, 0xF000, 12 )!= 6 )  // not followed by a Bcc?
{
```

```
        buf = new char[ 128 ];
        sprintf ( buf, "WARNING at address $%lX: You probably want to follow\n a CMP with a Bcc
            instruction (e.g. BEQ)", oldPC );
        helpStack.push( buf );
    }
}

void M68008::cmpm()
{
    unsigned char Ax = GET_INSTRBITS( 0xE00, 9 );
    unsigned char Ay = GET_INSTRBITS( 0x7, 0 );
    unsigned char Size = GET_INSTRBITS( 0xC0, 6 );
    DATA_REGISTER( Result );

    ccr.n = ccr.z = ccr.v = ccr.c = 0;

    unsigned long int oldPC = pc;

    switch ( Size )
    {
        case 0: // Byte
            Result.b = memory->GetByte( a[ Ax ].l )- memory->GetByte( a[ Ay ].l );
            SETNZB( Result.b )
            SUB_SETOC( memory->GetByte( a[ Ay ].l ), memory->GetByte( a[ Ax ].l ), Result.b, MSB8 );
            SUB_SETV( memory->GetByte( a[ Ay ].l ), memory->GetByte( a[ Ax ].l ), Result.b, MSB8 );

            a[ Ax ].l++;
            a[ Ay ].l++;

            break;
        case 1: // Word
            Result.w = memory->GetWord( a[ Ax ].l )- memory->GetWord( a[ Ay ].l );
            SETNZW( Result.w );
            SUB_SETOC( memory->GetWord( a[ Ay ].l ), memory->GetWord( a[ Ax ].l ), Result.w, MSB16 );
            SUB_SETV( memory->GetWord( a[ Ay ].l ), memory->GetWord( a[ Ax ].l ), Result.w, MSB16 );

            a[ Ax ].l += 2;
            a[ Ay ].l += 2;
            break;
        case 2: // Longword
            Result.l = memory->GetLongword( a[ Ax ].l )- memory->GetLongword( a[ Ay ].l );
            SETNZL( Result.l );
            SUB_SETOC( memory->GetLongword( a[ Ay ].l ), memory->GetLongword( a[ Ax ].l ), Result.l,
                MSB32 );
            SUB_SETV( memory->GetLongword( a[ Ay ].l ), memory->GetLongword( a[ Ax ].l ), Result.l,
                MSB32 );

            a[ Ax ].l += 4;
            a[ Ay ].l += 4;
            break;
    }
    pc += 2;
```

```
    // The following code should be put after the execution of the CMP instruction
    // It needs the PC to be updated to point to the instruction after the CMP instruction
    char *buf;
    if( GETBITS( pc, 0xF000, 12 )!= 6 )  // not followed by a Bcc?
    {
        buf = new char[ 128 ];
        sprintf( buf, "WARNING at address $%lX: You probably want to follow a CMP with a Bcc
            instruction (e.g. BEQ)", oldPC );
        helpStack.push( buf );
    }
}
```

1.5 TInstrA-C.cpp

The translation methods of the instructions starting with $A \cdots C$.

```
/*
    Motorola 68008 Simulator
    (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
    Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

    TInstrA-C.cpp -- Translation methods for instructions ABCD - CMPM
*/
#include <stdio.h>
#include "M68008.h"

#define GETBITS(addr, mask, shift) (memory->GetWord(addr) & mask) >> shift
#define GET_TINSTRBITS(mask, shift) (memory->GetWord( _address )& mask) >> shift
#define BYTE_MSB(byte) ( byte & 0x80 )>> 7
#define WORD_MSB(word) ( word & 0x8000 )>> 15
#define LONG_MSB(longw) ( longw & 0x80000000 )>> 31

unsigned long M68008::tS_abcd( unsigned long _address, char *buf )
{
    unsigned char RM = GETBITS( _address, 0x8, 3);
     unsigned char Rx = GETBITS( _address, 0xE00, 9);
     unsigned char Ry = GETBITS( _address, 0x7, 0);

    switch ( RM )
    {
        case 0: // Dy,Dx
            sprintf( buf, "abcd.b d%X, d%X", Ry, Rx );
            break;

        case 1: // -(Ay),-(Ax)
            sprintf( buf, "abcd.b -(a%X), -(a%X)", Ry, Rx );
            break;
    }
    _address += 2; // Update PC
    return _address;
}
```

```
unsigned long M68008::tS_add( unsigned long _address, char *buf )
{
    //
    // Alan Donnelly (12/04/2002)
    //

    //
    // This also handles ADDA ( same instruction signature )
    //
    // check immediate and (xxx).W/L
    //

    unsigned char Reg = GETBITS(_address, 0xE00, 9);
    unsigned char OpMode = GETBITS(_address, 0x1C0, 6);
    unsigned char EaMode = GETBITS(_address, 0x38, 3);
    unsigned char EaReg = GETBITS(_address, 0x7, 0);

    // (d16, An)
    signed short int Disp16;

    // Disp8(An, Xn)
    signed char Disp8;
    unsigned char Xn;
    unsigned char Wl;

    // get addressing mode and source (Ea) value
    // bool EaIsSource = GETBITS(pc, 0x100, 0) == 0;

    switch( OpMode )
    {
    case 0:  // BYTE (Source: Ea)

        switch( EaMode )
        {
        case 0:      // Dn.b
            sprintf ( buf, "add.b_d%X,d%X", EaReg, Reg);
            break;
        case 1:      // An.b (doesn't apply)
            tS_addx( _address, buf );
            break;
        case 2:      // (An).b
            sprintf ( buf, "add.b_(a%X),d%X", EaReg, Reg);
            break;
        case 3:      // (An)+.b
            sprintf ( buf, "add.b_(a%X)+,d%X", EaReg, Reg);
            break;
        case 4:      // -(An).b
            sprintf ( buf, "add.b_-(a%X),d%X", EaReg, Reg );
            break;
        case 5:      // (d16, An).b
            Disp16 = memory->GetWord( _address + 2 );
```

```
    if( decImm )sprintf( buf, "add.b %d(a%X),d%X", Disp16, EaReg, Reg );
    else sprintf( buf, "add.b $%X(a%X),d%X", Disp16, EaReg, Reg );

    // modify _address
    _address += 2;

    break;
case 6:     // Disp8(An, Xn).b
    // +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
    // | 0 |    Xn   | w/l | 0 | 0 | 0 |          Disp8               |
    // +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
    //

    Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
    Xn = GETBITS( _address + 2, 0x7000, 12 );
    Wl = GETBITS( _address + 2, 0x800, 11 );

    if( !Wl )       // word index
        if( decImm )sprintf( buf, "add.b %d(a%X,d%X.w),d%X", Disp8, EaReg, Xn, Reg );
        else sprintf( buf, "add.b $%X(a%X,d%X.w),d%X", Disp8, EaReg, Xn, Reg );
    else
        if( decImm )sprintf( buf, "add.b %d(a%X,d%X.l),d%X", Disp8, EaReg, Xn, Reg );
        else sprintf( buf, "add.b $%X(a%X,d%X.l),d%X", Disp8, EaReg, Xn, Reg );

    _address += 2;    // modify _address

    break;
case 7:
    switch( EaReg )
    {
        case 0:         // (www).W
            sprintf( buf, "add.b $%X,d%X", memory->GetWord( _address + 2 ), Reg );

            _address += 2; // modify _address
            break;
        case 1:         // (www).L
            sprintf( buf, "add.b $%lX,d%X", memory->GetLongword( _address + 2 ), Reg );

            _address += 4; // modify _address
            break;
        case 2:         // (d16, PC).b
            Disp16 = memory->GetWord( _address + 2 );
            if( decImm )sprintf( buf, "add.b %ld(PC),d%X", Disp16 + 2 + _address, Reg );
            else sprintf( buf, "add.b $%lX(PC),d%X", Disp16 + 2 + _address, Reg );

            _address += 2; // modify _address
            break;
        case 3:         // d8(PC, Xn)
            Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
            Xn = GETBITS( _address + 2, 0x7000, 12 );
            Wl = GETBITS( _address + 2, 0x800, 11 );
```

```
            if ( !Wl )   // word index
               if( decImm )sprintf( buf, "add.b␣%ld(PC,d%X.w),d%X", Disp8 + 2 + _address, Xn, Reg );
               else sprintf( buf, "add.b␣$%lX(PC,d%X.w),d%X", Disp8 + 2 + _address, Xn, Reg );
            else      // long index
               if( decImm )sprintf( buf, "add.b␣%ld(PC,d%X.l),d%X", Disp8 + 2 + _address, Xn, Reg );
               else sprintf( buf, "add.b␣$%lX(PC,d%X.l),d%X", Disp8 + 2 + _address, Xn, Reg );

            _address += 2; // modify _address
            break;
         case 4:          // #<data>
            if( decImm )sprintf( buf, "add.b␣#%d,d%X", memory->GetByte( _address + 3 ), Reg );
            else sprintf( buf, "add.b␣#$%X,d%X", memory->GetByte( _address + 3 ), Reg );
            _address += 2;    // modify _address
            break;
         case 5:
         case 6:
         case 7:
            tS_addx( _address, buf );
            break;
      }
      break;
   }
break;

case 1:
   switch( EaMode )
   {
   // WORD (Source: Ea)
   case 0:      // Dn.w
      sprintf( buf, "add.w␣d%X,d%X", EaReg, Reg );
      break;
   case 1:      // An.w
      sprintf( buf, "add.w␣a%X,d%X", EaReg, Reg );
      break;
   case 2:      // (An).w
      sprintf( buf, "add.w␣(a%X),d%X", EaReg, Reg );
      break;
   case 3:      // (An)+.w
      sprintf( buf, "add.w␣(a%X)+,d%X", EaReg, Reg );
      break;
   case 4:      // -(An).w
      sprintf( buf, "add.w␣-(a%X),d%X", EaReg, Reg );
      break;
   case 5:  // (d16, An).w
      Disp16 = memory->GetWord( _address + 2 );
      if( decImm )sprintf( buf, "add.w␣%d(a%X),d%X", Disp16, EaReg, Reg );
      else sprintf( buf, "add.w␣$%X(a%X),d%X", Disp16, EaReg, Reg );

      _address += 2;    // modify _address
      break;
   case 6:      // Disp8(An, Xn).w
```

```
    Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
    Xn = GETBITS( _address + 2, 0x7000, 12 );
    Wl = GETBITS( _address + 2, 0x800, 11 );

    if ( !Wl )      // word index
      if ( decImm )sprintf( buf, "add.w %d(a%X,d%X.w),d%X", Disp8, EaReg, Xn, Reg );
      else sprintf( buf, "add.w $%X(a%X,d%X.w),d%X", Disp8, EaReg, Xn, Reg );
    else
      if ( decImm )sprintf( buf, "add.w %d(a%X,d%X.l),d%X", Disp8, EaReg, Xn, Reg );
      else sprintf( buf, "add.w $%X(a%X,d%X.l),d%X", Disp8, EaReg, Xn, Reg );

    _address += 2;    // modify _address
    break;
  case 7:
    switch( EaReg )
    {
      case 0:         // (www).W
        sprintf( buf, "add.w $%X,d%X", memory->GetWord( _address + 2 ), Reg );

        _address += 2; // modify _address
        break;
      case 1:         // (www).L
        sprintf( buf, "add.w $%lX,d%X", memory->GetLongword( _address + 2 ), Reg );

        _address += 4; // modify _address
        break;
      case 2:         // (d16, PC).w
        Disp16 = memory->GetWord( _address + 2 );
        if ( decImm )sprintf( buf, "add.w %ld(PC),d%X", Disp16 + 2 + _address, Reg );
        else sprintf( buf, "add.w $%lX(PC),d%X", Disp16 + 2 + _address, Reg );

        _address += 2; // modify _address
        break;
      case 3:         // d8(PC, Xn)
        Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
        Xn = GETBITS( _address + 2, 0x7000, 12 );
        Wl = GETBITS( _address + 2, 0x800, 11 );

        if ( !Wl )    // word index
          if ( decImm )sprintf( buf, "add.w %ld(PC,d%X.w),d%X", Disp8 + 2 + _address, Xn, Reg );
          else sprintf( buf, "add.w $%lX(PC,d%X.w),d%X", Disp8 + 2 + _address, Xn, Reg );
        else      // long index
          if ( decImm )sprintf( buf, "add.w %ld(PC,d%X.l),d%X", Disp8 + 2 + _address, Xn, Reg );
          else sprintf( buf, "add.w $%lX(PC,d%X.l),d%X", Disp8 + 2 + _address, Xn, Reg );

        _address += 2; // modify _address
        break;
      case 4:         // #<data>
        if ( decImm )sprintf( buf, "add.w #%d,d%X", memory->GetWord( _address + 2), Reg );
        else sprintf( buf, "add.w #$%X,d%X", memory->GetWord( _address + 2), Reg );

        _address += 2; // modify _address
```

```
                break;
            case 5:
            case 6:
            case 7:
                tS_addx( _address, buf );
                break;
        }
        break;
    }

    break;

case 2:     // LONGWORD (Source: Ea)

    switch( EaMode )
    {
    case 0:     // Dn.l
        sprintf( buf, "add.l␣d%X,d%X", EaReg, Reg );
        break;
    case 1:     // An.l
        sprintf( buf, "add.l␣a%X,d%X", EaReg, Reg );
        break;
    case 2:     // (An).l
        sprintf( buf, "add.l␣(a%X),d%X", EaReg, Reg );
        break;
    case 3:     // (An)+.l
        sprintf( buf, "add.l␣(a%X)+,d%X", EaReg, Reg );
        break;
    case 4:     // -(An).l
        sprintf( buf, "add.l␣-(a%X),d%X", EaReg, Reg );
        break;
    case 5:     // (d16, An).l
        Disp16 = memory->GetWord( _address + 2 );
        if( decImm )sprintf( buf, "add.l␣%d(a%X),d%X", Disp16, EaReg, Reg );
        else sprintf( buf, "add.l␣$%X(a%X),d%X", Disp16, EaReg, Reg );

        _address += 2; // modify _address
        break;
    case 6:     // Disp8(An, Xn).l
        Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
        Xn = GETBITS( _address + 2, 0x7000, 12 );
        Wl = GETBITS( _address + 2, 0x800, 11 );

        if ( !Wl )   // word index
            if( decImm )sprintf( buf, "add.l␣%d(a%X,d%X.w),d%X", Disp8, EaReg, Xn, Reg );
            else sprintf( buf, "add.l␣$%X(a%X,d%X.w),d%X", Disp8, EaReg, Xn, Reg );
        else        // long index
            if( decImm )sprintf( buf, "add.l␣%d(a%X,d%X.l),d%X", Disp8, EaReg, Xn, Reg );
            else sprintf( buf, "add.l␣$%X(a%X,d%X.l),d%X", Disp8, EaReg, Xn, Reg );

        _address += 2; // modify _address
        break;
```

```
    case 7:
        switch( EaReg )
        {
            case 0:          // (www).W
                sprintf ( buf, " add.l $%X,d%X", memory->GetWord( _address + 2 ), Reg );

                _address += 2; // modify _address
                break;
            case 1:          // (www).L
                sprintf ( buf, " add.l $%X,d%X", memory->GetWord( _address + 2 ), Reg );

                _address += 4; // modify _address
                break;
            case 2:          // (d16, PC).l
                Disp16 = memory->GetWord( _address + 2 );
                if ( decImm )sprintf ( buf, " add.l %ld(PC),d%X", Disp16 + 2 + _address, Reg );
                else  sprintf ( buf, " add.l $%lX(PC),d%X", Disp16 + 2 + _address, Reg );

                _address += 2; // modify _address
                break;
            case 3:          // d8(PC, Xn)
                Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
                Xn = GETBITS( _address + 2, 0x7000, 12 );
                Wl = GETBITS( _address + 2, 0x800, 11 );

                if ( !Wl )    // word index
                    if ( decImm )sprintf ( buf, " add.l %ld(PC,d%X.w),d%X", Disp8 + 2 + _address, Xn, Reg );
                    else  sprintf ( buf, " add.l $%lX(PC,d%X.w),d%X", Disp8 + 2 + _address, Xn, Reg );
                else       // long index
                    if ( decImm )sprintf ( buf, " add.l %ld(PC,d%X.l),d%X", Disp8 + 2 + _address, Xn, Reg );
                    else  sprintf ( buf, " add.l $%lX(PC,d%X.l),d%X", Disp8 + 2 + _address, Xn, Reg );

                _address += 2; // modify _address
                break;
            case 4:          // #<data>
                if ( decImm )sprintf ( buf, " add.l #%ld,d%X", memory->GetLongword( _address + 2 ), Reg );
                else  sprintf ( buf, " add.l #$%lX,d%X", memory->GetLongword( _address + 2 ), Reg );

                _address += 4; // modify _address
                break;
            case 5:
            case 6:
            case 7:
                tS_addx( _address, buf );
                break;
        }

    break;
    }
break;

case 4:      // BYTE (Source: Dn)
```

```
switch( EaMode )
{
case 0:     // Dn.b (n/a for dest = Ea)
case 1:     // An.b (n/a for dest = Ea)
   tS_addx( _address, buf );
   break;
case 2:     // (An).b
   sprintf( buf, "add.b␣d%X,(a%X)", Reg, EaReg );
   break;
case 3:     // (An)+.b
   sprintf( buf, "add.b␣d%X,(a%X)+", Reg, EaReg );
   break;
case 4:     // -(An).b
   sprintf( buf, "add.b␣d%X,-(a%X)", Reg, EaReg );
   break;
case 5:     // (d16, An).b
   Disp16 = memory->GetWord( _address + 2 );

   if( decImm )sprintf( buf, "add.b␣d%X,%d(a%X)", Reg, Disp16, EaReg );
   else  sprintf( buf, "add.b␣d%X,$%X(a%X)", Reg, Disp16, EaReg );

   _address += 2; // modify _address
   break;
case 6:     // Disp8(An, Xn).b
   Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
   Xn = GETBITS( _address + 2, 0x7000, 12 );
   Wl = GETBITS( _address + 2, 0x800, 11 );

   if( !Wl )   // word index
      if( decImm )sprintf( buf, "add.b␣d%X,%d(a%X,d%X.w)", Reg, Disp8, EaReg, Xn );
      else  sprintf( buf, "add.b␣d%X,$%X(a%X,d%X.w)", Reg, Disp8, EaReg, Xn );
   else     // long index
      if( decImm )sprintf( buf, "add.b␣d%X,%d(a%X,d%X.l)", Reg, Disp8, EaReg, Xn );
      else  sprintf( buf, "add.b␣d%X,$%X(a%X,d%X.l)", Reg, Disp8, EaReg, Xn );

   _address += 2; // modify _address
   break;
case 7:
   switch( EaReg )
   {
      case 0:          // (www).W
         sprintf( buf, "add.b␣d%X,$%X", Reg, memory->GetWord( _address + 2 ));

         _address += 2; // modify _address
         break;
      case 1:          // (www).L
         sprintf( buf, "add.b␣d%X,$%lX", Reg, memory->GetLongword( _address + 2 ));

         _address += 4; // modify _address
         break;
      case 2:          // (d16, PC).b
```

```
            case 3:         // d8(PC, Xn)
            case 4:         // #<data>
            case 5:
            case 6:
            case 7:
                tS_addx( _address, buf );
                break;
        }
        break;
    }
    break;

case 5:     // WORD (Source: Dn)
    switch( EaMode )
    {
    case 0:     // Dn.w (n/a for dest = Ea)
    case 1:     // An.w (n/a for dest = Ea)
        tS_addx( _address, buf );
        break;
    case 2:     // (An).w
        sprintf( buf, "add.w_d%X,(a%X)", Reg, EaReg );
        break;
    case 3:     // (An)+.w
        sprintf( buf, "add.w_d%X,(a%X)+", Reg, EaReg );
        break;
    case 4:     // -(An).w
        sprintf( buf, "add.w_d%X,-(a%X)", Reg, EaReg );
        break;
    case 5:     // (d16, An).w
        Disp16 = memory->GetWord( _address + 2 );
        if( decImm )sprintf( buf, "add.w_d%X,%d(a%X)", Reg, Disp16, EaReg );
        else  sprintf( buf, "add.w_d%X,$%X(a%X)", Reg, Disp16, EaReg );

        _address += 2; // modify _address
        break;
    case 6:     // Disp8(An, Xn).w
        Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
        Xn = GETBITS( _address + 2, 0x7000, 12 );
        Wl = GETBITS( _address + 2, 0x800, 11 );

        if ( !Wl )   // word index
            if( decImm )sprintf( buf, "add.w_d%X,%d(a%X,d%X.w)", Reg, Disp8, EaReg, Xn );
            else  sprintf( buf, "add.w_d%X,$%X(a%X,d%X.w)", Reg, Disp8, EaReg, Xn );
        else     // long index
            if( decImm )sprintf( buf, "add.w_d%X,%d(a%X,d%X.l)", Reg, Disp8, EaReg, Xn );
            else  sprintf( buf, "add.w_d%X,$%X(a%X,d%X.l)", Reg, Disp8, EaReg, Xn );

        _address += 2; // modify _address
        break;
    case 7:
        switch( EaReg )
        {
```

```
            case 0:         // (www).W
                sprintf ( buf, "add.w d%X,$%X", Reg, memory->GetWord( _address + 2) );

                _address += 2; // modify _address
                break;
            case 1:         // (www).L
                sprintf ( buf, "add.w d%X,$%lX", Reg, memory->GetLongword( _address + 2) );

                _address += 4; // modify _address
                break;
            case 2:         // (d16, PC).w (n/a for dest = Ea)
            case 3:         // d8(PC, Xn) (n/a for dest = Ea)
            case 4:         // #<data>   (n/a for dest = Ea)
            case 5:
            case 6:
            case 7:
                tS_addx( _address, buf );
                break;
        }
        break;
    }

    break;

case 6:     // LONGWORD (Source: Dn)
    switch( EaMode )
    {
        case 0:     // Dn.l (n/a for dest = Ea)
        case 1:     // An.l (n/a for dest = Ea)
            tS_addx( _address, buf );
            break;
        case 2:     // (An).l
            sprintf ( buf, "add.l d%X,(a%X)", Reg, EaReg );
            break;
        case 3:     // (An)+.l
            sprintf ( buf, "add.l d%X,(a%X)+", Reg, EaReg );
            break;
        case 4:     // -(An).l
            sprintf ( buf, "add.l d%X,-(a%X)", Reg, EaReg );
            break;
        case 5:     // (d16, An).l
            Disp16 = memory->GetWord( _address + 2);
            if ( decImm )sprintf ( buf, "add.l d%X,%d(a%X)", Reg, Disp16, EaReg );
            else  sprintf ( buf, "add.l d%X,$%X(a%X)", Reg, Disp16, EaReg );

            _address += 2; // modify _address
            break;
        case 6:     // Disp8(An, Xn).l
            Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
            Xn = GETBITS( _address + 2, 0x7000, 12 );
            Wl = GETBITS( _address + 2, 0x800, 11 );
```

```
            if ( !Wl )    // word index
                if ( decImm )sprintf ( buf, "add.l␣d%X,%d(a%X,d%X.w)", Reg, Disp8, EaReg, Xn );
                else  sprintf ( buf, "add.l␣d%X,$%X(a%X,d%X.w)", Reg, Disp8, EaReg, Xn );
            else      // long index
                if ( decImm )sprintf ( buf, "add.l␣d%X,%d(a%X,d%X.l)", Reg, Disp8, EaReg, Xn );
                else  sprintf ( buf, "add.l␣d%X,$%X(a%X,d%X.l)", Reg, Disp8, EaReg, Xn );

            _address += 2; // modify _address
            break;
        case 7:
            switch( EaReg )
            {
                case 0:         // (www).W
                    sprintf ( buf, "add.l␣d%X,$%X", Reg, memory->GetWord( _address + 2 ));

                    _address += 2; // modify _address
                    break;
                case 1:         // (www).L
                    sprintf ( buf, "add.l␣d%X,$%lX", Reg, memory->GetLongword( _address + 2 ));

                    _address += 4; // modify _address
                    break;
                case 2:         // (d16, PC).l (n/a for dest = Ea)
                case 3:         // d8(PC, Xn) (n/a for dest = Ea)
                case 4:         // #<data>    (n/a for dest = Ea)
                case 5:
                case 6:
                case 7:
                    tS_addx( _address, buf );
                    break;
            }
            break;
        }

        break;
case 3:
    //
    // ADDA: Add to address register
    //

    // WORD

    switch( EaMode )
    {
    case 0:     // Dn.w
        sprintf ( buf, "adda.w␣d%X,a%X", EaReg, Reg );
        break;
    case 1:     // An.w
        sprintf ( buf, "adda.w␣a%X,a%X", EaReg, Reg );
        break;
    case 2:     // (An).w
        sprintf ( buf, "adda.w␣(a%X),a%X", EaReg, Reg );
```

```
    break;
case 3:     // (An)+.w
    sprintf( buf, "adda.w (a%X)+,a%X", EaReg, Reg );
    break;
case 4:     // -(An).w
    sprintf( buf, "adda.w -(a%X),a%X", EaReg, Reg );
    break;
case 5:     // (d16, An).w
    Disp16 = memory->GetWord( _address + 2 );
    if( decImm )sprintf( buf, "adda.w %d(a%X),a%X", Disp16, EaReg, Reg );
    else  sprintf( buf, "adda.w $%X(a%X),a%X", Disp16, EaReg, Reg );

    _address += 2;
    break;
case 6:     // Disp8(An, Xn).w
    Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
    Xn = GETBITS( _address + 2, 0x7000, 12 );
    Wl = GETBITS( _address + 2, 0x800, 11 );

    if ( !Wl )   // word index
        if( decImm )sprintf( buf, "adda.w %d(a%X,d%X.w),a%X", Disp8, EaReg, Xn, Reg );
        else  sprintf( buf, "adda.w $%X(a%X,d%X.w),a%X", Disp8, EaReg, Xn, Reg );
    else     // long index
        if( decImm )sprintf( buf, "adda.w %d(a%X,d%X.l),a%X", Disp8, EaReg, Xn, Reg );
        else  sprintf( buf, "adda.w $%X(a%X,d%X.l),a%X", Disp8, EaReg, Xn, Reg );

    _address += 2; // modify _address
    break;
case 7:
    switch( EaReg )
    {
        case 0:          // (www).W
            sprintf( buf, "adda.w $%X,a%X", memory->GetWord( _address + 2 ), Reg );

            _address += 2; // modify _address
            break;
        case 1:          // (www).L
            sprintf( buf, "adda.w $%lX,a%X", memory->GetLongword( _address + 2 ), Reg );

            _address += 4; // modify _address
            break;
        case 2:          // (d16, PC).w
            Disp16 = memory->GetWord( _address + 2 );
            if( decImm )sprintf( buf, "adda.w %ld(PC),a%X", Disp16 + 2 + _address, Reg );
            else  sprintf( buf, "adda.w $%lX(PC),a%X", Disp16 + 2 + _address, Reg );

            _address += 2; // modify _address
            break;
        case 3:          // d8(PC, Xn)
            Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
            Xn = GETBITS( _address + 2, 0x7000, 12 );
            Wl = GETBITS( _address + 2, 0x800, 11 );
```

```
            if ( !Wl )    // word index
               if ( decImm )sprintf( buf, "adda.w %ld(PC,d%X.w),a%X", Disp8 + 2 + _address, Xn, Reg );
               else  sprintf( buf, "adda.w $%lX(PC,d%X.w),a%X", Disp8 + 2 + _address, Xn, Reg );
            else      // long index
               if ( decImm )sprintf( buf, "adda.w %ld(PC,d%X.l),a%X", Disp8 + 2 + _address, Xn, Reg );
               else  sprintf( buf, "adda.w $%lX(PC,d%X.l),a%X", Disp8 + 2 + _address, Xn, Reg );

            _address += 2; // modify _address
            break;
         case 4:          // #<data>
            if ( decImm )sprintf( buf, "adda.w #%d,a%X", memory->GetWord( _address + 2 ), Reg );
            else  sprintf( buf, "adda.w #$%X,a%X", memory->GetWord( _address + 2 ), Reg );

            _address += 2; // modify _address
            break;
         case 5:
         case 6:
         case 7:
            tS_addx( _address, buf );
            break;
      }
      break;
   }

   break;

case 7:

   //
   // ADDA: LONGWORD (Source: Ea)
   //


   switch( EaMode )
   {
   case 0:      // Dn.l
      sprintf( buf, "adda.l d%X,a%X", EaReg, Reg );
      break;
   case 1:      // An.l
      sprintf( buf, "adda.l a%X,a%X", EaReg, Reg );
      break;
   case 2:      // (An).l
      sprintf( buf, "adda.l (a%X),a%X", EaReg, Reg );
      break;
   case 3:      // (An)+.l
      sprintf( buf, "adda.l (a%X)+,a%X", EaReg, Reg );
      break;
   case 4:      // -(An).l
      sprintf( buf, "adda.l -(a%X),a%X", EaReg, Reg );
      break;
   case 5:      // (d16, An).l
```

```
    Disp16 = memory->GetWord( _address + 2 );
    if ( decImm )sprintf( buf, "adda.l %d(a%X),a%X", Disp16, EaReg, Reg );
    else sprintf( buf, "adda.l $%X(a%X),a%X", Disp16, EaReg, Reg );

    _address += 2; // modify _address
    break;
  case 6:     // Disp8(An, Xn).l
    Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
    Xn = GETBITS( _address + 2, 0x7000, 12 );
    Wl = GETBITS( _address + 2, 0x800, 11 );

    if ( !Wl )   // word index
      if ( decImm )sprintf( buf, "adda.l %d(a%X,d%X.w),a%X", Disp8, EaReg, Xn, Reg );
      else sprintf( buf, "adda.l $%X(a%X,d%X.w),a%X", Disp8, EaReg, Xn, Reg );
    else     // long index
      if ( decImm )sprintf( buf, "adda.l %d(a%X,d%X.l),a%X", Disp8, EaReg, Xn, Reg );
      else sprintf( buf, "adda.l $%X(a%X,d%X.l),a%X", Disp8, EaReg, Xn, Reg );

    _address += 2; // modify _address
    break;
  case 7:
    switch( EaReg )
    {
      case 0:         // (www).W
        sprintf( buf, "adda.l $%X,a%X", memory->GetWord( _address + 2 ), Reg );

        _address += 2; // modify _address
        break;
      case 1:         // (www).L
        sprintf( buf, "adda.l $%lX,a%X", memory->GetLongword( _address + 2 ), Reg );

        _address += 4; // modify _address
        break;
      case 2:         // (d16, PC).l
        Disp16 = memory->GetWord( _address + 2 );
        if ( decImm )sprintf( buf, "adda.l %ld(PC),a%X", Disp16 + 2 + _address, Reg );
        else sprintf( buf, "adda.l $%lX(PC),a%X", Disp16 + 2 + _address, Reg );

        _address += 2; // modify _address
        break;
      case 3:         // d8(PC, Xn)
        Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
        Xn = GETBITS( _address + 2, 0x7000, 12 );
        Wl = GETBITS( _address + 2, 0x800, 11 );

        if ( !Wl )   // word index
          if ( decImm )sprintf( buf, "adda.l %ld(PC,d%X.w),a%X", Disp8 + 2 + _address, Xn, Reg );
          else sprintf( buf, "adda.l $%lX(PC,d%X.w),a%X", Disp8 + 2 + _address, Xn, Reg );
        else     // long index
          if ( decImm )sprintf( buf, "adda.l %ld(PC,d%X.l),a%X", Disp8 + 2 + _address, Xn, Reg );
          else sprintf( buf, "adda.l $%lX(PC,d%X.l),a%X", Disp8 + 2 + _address, Xn, Reg );
```

```
                _address += 2; // modify _address
                break;
            case 4:          // #<data>
                if( decImm )sprintf( buf, "adda.l #%ld,a%X", memory->GetLongword( _address + 2 ), Reg );
                else  sprintf( buf, "adda.l #$%lX,a%X", memory->GetLongword( _address + 2 ), Reg );

                _address += 4; // modify _address
                break;
            case 5:
            case 6:
            case 7:
                tS_addx( _address, buf );
                break;
        }
        break;
    }
}

// Update the _address
_address += 2;

return _address;
}

unsigned long M68008::tS_addi( unsigned long _address, char *buf )
{
//
// Alan Donnelly (13/04/02)
//

unsigned char Size = GET_TINSTRBITS(0xC0, 6);
unsigned char EaMode = GET_TINSTRBITS(0x38, 3);
unsigned char EaReg = GET_TINSTRBITS(0x7, 0);

// (d16, An)
signed short int Disp16;

// Disp8(An, Xn)
signed char Disp8;
unsigned char Xn;
unsigned char Wl;

DATA_REGISTER( Data );

switch( Size )
{
    case 0:      // BYTE

        // Uses the LSByte of the word following the  instruction
        Data.b = memory->GetWord( _address + 2 )& 0xFF;
        _address += 2;
```

```
switch( EaMode )
{
   case 0:      // Dn.b
      if( decImm )sprintf( buf, "addi.b␣#%d,d%X", Data.b, EaReg );
      else sprintf( buf, "addi.b␣#$%X,d%X", Data.b, EaReg );
      break;
   case 1:      // An.b (n/a)
      illegal();
      break;
   case 2:      // (An).b
      if( decImm )sprintf( buf, "addi.b␣#%d,(a%X)", Data.b, EaReg );
      else sprintf( buf, "addi.b␣#$%X,(a%X)", Data.b, EaReg );
      break;
   case 3:      // (An)+.b
      if( decImm )sprintf( buf, "addi.b␣#%d,(a%X)+", Data.b, EaReg );
      else sprintf( buf, "addi.b␣#$%X,(a%X)+", Data.b, EaReg );
      break;
   case 4:      // --(An).b
      if( decImm )sprintf( buf, "addi.b␣#%d,-(a%X)", Data.b, EaReg );
      else sprintf( buf, "addi.b␣#$%X,-(a%X)", Data.b, EaReg );
      break;
   case 5:      // Disp16(An).b
      Disp16 = memory->GetWord( _address + 2 );
      if( decImm )sprintf( buf, "addi.b␣#%d,%d(a%X)", Data.b, Disp16, EaReg );
      else sprintf( buf, "addi.b␣#$%X,$%X(a%X)", Data.b, Disp16, EaReg );

      _address += 2;
      break;
   case 6:      // Disp8(An, Xn).b
      Disp8 = memory->GetByte( _address + 3 );
      Xn = GETBITS( _address + 2, 0x7000, 12 );
      Wl = GETBITS( _address + 2, 0x800, 11 );

      if ( !Wl )
         if( decImm )sprintf( buf, "addi.b␣#%d,%d(a%X,d%X.w)", Data.b, Disp8, EaReg, Xn );
         else sprintf( buf, "addi.b␣#$%X,$%X(a%X,d%X.w)", Data.b, Disp8, EaReg, Xn );
      else
         if( decImm )sprintf( buf, "addi.b␣#%d,%d(a%X,d%X.l)", Data.b, Disp8, EaReg, Xn );
         else sprintf( buf, "addi.b␣#$%X,$%X(a%X,d%X.l)", Data.b, Disp8, EaReg, Xn );

      _address += 2;
      break;
   case 7:
      switch ( EaReg )
      {
         case 0: // (xxx).W.b
            if( decImm )sprintf( buf, "addi.b␣#%d,$%X", Data.b, memory->GetWord( _address
                  + 2 ));
            else sprintf( buf, "addi.b␣#$%X,$%X", Data.b, memory->GetWord( _address + 2 ));
            _address += 2;
            break;
         case 1: // (xxx).L.b
```

```
                if( decImm )sprintf( buf, "addi.b #%d,$%lX", Data.b, memory->GetLongword(
                    _address + 2 ));
                else sprintf( buf, "addi.b #$%X,$%lX", Data.b, memory->GetLongword( _address + 2 )
                    );
                _address += 4;
                break;
            default:
                illegal ();
                break;
        }
        break;
    }

    break;

case 1:     // WORD
    Data.w = memory->GetWord( _address + 2 );
    _address += 2;

    switch( EaMode )
    {
        case 0:     // Dn.w
            if( decImm )sprintf( buf, "addi.w #%d,d%X", Data.w, EaReg );
            else sprintf( buf, "addi.w #$%X,d%X", Data.w, EaReg );
            break;
        case 1:     // An.w
            if( decImm )sprintf( buf, "addi.w #%d,a%X", Data.w, EaReg );
            else sprintf( buf, "addi.w #$%X,a%X", Data.w, EaReg );
            break;
        case 2:     // (An).w
            if( decImm )sprintf( buf, "addi.w #%d,(a%X)", Data.w, EaReg );
            else sprintf( buf, "addi.w #$%X,(a%X)", Data.w, EaReg );
            break;
        case 3:     // (An)+.w
            if( decImm )sprintf( buf, "addi.w #%d,(a%X)+", Data.w, EaReg );
            else sprintf( buf, "addi.w #$%X,(a%X)+", Data.w, EaReg );
            break;
        case 4:     // -(An).w
            if( decImm )sprintf( buf, "addi.w #%d,-(a%X)", Data.w, EaReg );
            else sprintf( buf, "addi.w #$%X,-(a%X)", Data.w, EaReg );
            break;
        case 5:     // Disp16(An).w
            Disp16 = memory->GetWord( _address + 2 );
            if( decImm )sprintf( buf, "addi.w #%d,%d(a%X)", Data.w, Disp16, EaReg );
            else sprintf( buf, "addi.w #$%X,$%X(a%X)", Data.w, Disp16, EaReg );

            _address += 2;
            break;
        case 6:     // Disp8(An, Xn).w
            Disp8 = memory->GetByte( _address + 3 );
            Xn = GETBITS( _address + 2, 0x7000, 12 );
            Wl = GETBITS( _address + 2, 0x800, 11 );
```

```
            if ( !Wl )
                if( decImm )sprintf( buf, "addi.w #%d,%d(a%X,d%X.w)", Data.w, Disp8, EaReg, Xn );
                else sprintf( buf, "addi.w #$%X,$%X(a%X,d%X.w)", Data.w, Disp8, EaReg, Xn );
            else
                if( decImm )sprintf( buf, "addi.w #%d,%d(a%X,d%X.l)", Data.w, Disp8, EaReg, Xn );
                else sprintf( buf, "addi.w #$%X,$%X(a%X,d%X.l)", Data.w, Disp8, EaReg, Xn );

            _address += 2;
            break;
        case 7:
            switch ( EaReg )
            {
                case 0: // (xxx).W.w
                    if( decImm )sprintf( buf, "addi.w #%d,$%X", Data.w, memory->GetWord( _address
                        + 2 ));
                    else sprintf( buf, "addi.w #$%X,$%X", Data.w, memory->GetWord( _address + 2 ));

                    _address += 2;
                    break;
                case 1: // (xxx).L.w
                    if( decImm )sprintf( buf, "addi.w #%d,$%lX", Data.w, memory->GetLongword(
                        _address + 2 ));
                    else sprintf( buf, "addi.w #$%X,$%lX", Data.w, memory->GetLongword( _address
                        + 2 ));

                    _address += 4;
                    break;
                default:
                    illegal ();
                    break;
            }
            break;
    }
    break;

    case 2: // LONGWORD
    Data.l = memory->GetLongword( _address + 2 );
    _address += 4;

    switch( EaMode )
    {
        case 0:     // Dn.l
            if( decImm )sprintf( buf, "addi.l #%ld,d%X", Data.l, EaReg );
            else sprintf( buf, "addi.l #$%lX,d%X", Data.l, EaReg );
            break;
        case 1:     // An.l
            if( decImm )sprintf( buf, "addi.l #%ld,a%X", Data.l, EaReg );
            else sprintf( buf, "addi.l #$%lX,a%X", Data.l, EaReg );
            break;
        case 2:     // (An).l
            if( decImm )sprintf( buf, "addi.l #%ld,(a%X)", Data.l, EaReg );
            else sprintf( buf, "addi.l #$%lX,(a%X)", Data.l, EaReg );
```

```
    break;
case 3:     // (An)+.l
    if( decImm )sprintf( buf, "addi.l #%ld,(a%X)+", Data.l, EaReg );
    else sprintf( buf, "addi.l #$%lX,(a%X)+", Data.l, EaReg );
    break;
case 4:     // --(An).l
    if( decImm )sprintf( buf, "addi.l #%ld,-(a%X)", Data.l, EaReg );
    else sprintf( buf, "addi.l #$%lX,-(a%X)", Data.l, EaReg );
    break;
case 5:     // Disp16(An).l
    Disp16 = memory->GetWord( _address + 2 );
    if( decImm )sprintf( buf, "addi.l #%ld,%d(a%X)", Data.l, Disp16, EaReg );
    else sprintf( buf, "addi.l #$%lX,$%X(a%X)", Data.l, Disp16, EaReg );

    _address += 2;
    break;
case 6:     // Disp8(An, Xn).l
    Disp8 = memory->GetByte( _address + 3 );
    Xn = GETBITS( _address + 2, 0x7000, 12 );
    Wl = GETBITS( _address + 2, 0x800, 11 );
    if ( !Wl )
        if( decImm )sprintf( buf, "addi.l #%ld,%d(a%X,d%X.w)", Data.l, Disp8, EaReg, Xn );
        else sprintf( buf, "addi.l #$%lX,$%X(a%X,d%X.w)", Data.l, Disp8, EaReg, Xn );
    else
        if( decImm )sprintf( buf, "addi.l #%ld,%d(a%X,d%X.l)", Data.l, Disp8, EaReg, Xn );
        else sprintf( buf, "addi.l #$%lX,$%X(a%X,d%X.l)", Data.l, Disp8, EaReg, Xn );

    _address += 2;
    break;
case 7:
    switch ( EaReg )
    {
        case 0: // (xxx).W.l
            if( decImm )sprintf( buf, "addi.l #%ld,$%X", Data.l, memory->GetWord( _address
                + 2) );
            else sprintf( buf, "addi.l #$%lX,$%X", Data.l, memory->GetWord( _address + 2) );

            _address += 2;
            break;
        case 1: // (xxx).L.l
            if( decImm )sprintf( buf, "addi.l #%ld,$%lX", Data.l, memory->GetLongword(
                _address + 2) );
            else sprintf( buf, "addi.l #$%lX,$%lX", Data.l, memory->GetLongword( _address + 2)
                );

            _address += 4;
            break;
        default:
            illegal ();
            break;
    }
    break;
```

```
        }
        break;
    }

    // Update _address
    _address += 2;

    return _address;
}

unsigned long M68008::tS_addq( unsigned long _address, char *buf )
{
    //
    // Alan Donnelly (14/04/02)
    //

    unsigned char Data = GET_TINSTRBITS(0xE00, 9);
    unsigned char Size = GET_TINSTRBITS(0xC0, 6);
    unsigned char EaMode = GET_TINSTRBITS(0x38, 3);
    unsigned char EaReg = GET_TINSTRBITS(0x7, 0);

    // (d16, An)
    signed short int Disp16;

    // Disp8(An, Xn)
    signed char Disp8;
    unsigned char Xn;
    unsigned char Wl;

    // Check for Data == (this is 8)
    if( Data == 0 )Data = 8;

    switch ( Size )
    {
        // BYTE (Source: Dn)
    case 0:
        switch( EaMode )
        {
        case 0:      // Dn.b
            if( decImm )sprintf( buf, "addq.b #%d,d%X", Data, EaReg );
            else sprintf( buf, "addq.b #$%X,d%X", Data, EaReg );
            break;
        case 1:      // An.b (n/a for dest = Ea)
            illegal ();
            break;
        case 2:      // (An).b
            if( decImm )sprintf( buf, "addq.b #%d,(a%X)", Data, EaReg );
            else sprintf( buf, "addq.b #$%X,(a%X)", Data, EaReg );
            break;
        case 3:      // (An)+.b
            if( decImm )sprintf( buf, "addq.b #%d,(a%X)+", Data, EaReg );
            else sprintf( buf, "addq.b #$%X,(a%X)+", Data, EaReg );
```

```
    break;
case 4:     // -(An).b
    if( decImm )sprintf( buf, "addq.b #%d,-(a%X)", Data, EaReg );
    else sprintf( buf, "addq.b #$%X,-(a%X)", Data, EaReg );
    break;
case 5:     // (d16, An).b
    Disp16 = memory->GetWord( _address + 2 );
    if( decImm )sprintf( buf, "addq.b #%d,%d(a%X)", Data, Disp16, EaReg );
    else sprintf( buf, "addq.b #$%X,$%X(a%X)", Data, Disp16, EaReg );

    _address += 2;
    break;
case 6:     // Disp8(An, Xn).b

    Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
    Xn = GETBITS( _address + 2, 0x7000, 12 );
    Wl = GETBITS( _address + 2, 0x800, 11 );

    if ( !Wl )      // word index
        if( decImm )sprintf( buf, "addq.b #%d,%d(a%X,d%X.w)", Data, Disp8, EaReg, Xn );
        else sprintf( buf, "addq.b #$%X,$%X(a%X,d%X.w)", Data, Disp8, EaReg, Xn );
    else
        if( decImm )sprintf( buf, "addq.b #%d,%d(a%X,d%X.l)", Data, Disp8, EaReg, Xn );
        else sprintf( buf, "addq.b #$%X,$%X(a%X,d%X.l)", Data, Disp8, EaReg, Xn );

    // Update PC
    _address += 2;
    break;
case 7:
    switch( EaReg )
    {
        case 0:         // (www).W
            if( decImm )sprintf( buf, "addq.b #%d,$%X", Data, memory->GetWord( _address + 2 ));
            else sprintf( buf, "addq.b #$%X,$%X", Data, memory->GetWord( _address + 2 ));

            _address += 2;
            break;
        case 1:         // (www).L
            if( decImm )sprintf( buf, "addq.b #%d,$%lX", Data, memory->GetLongword( _address + 2 ))
                ;
            else sprintf( buf, "addq.b #$%X,$%lX", Data, memory->GetLongword( _address + 2 ));

            _address += 4;
            break;
        case 2:         // (d16, PC).b (n/a)
        case 3:         // d8(PC, Xn) (n/a)
        case 4:         // #<data>   (n/a)
        case 5:
        case 6:
        case 7:
            // illegal ();
            break;
```

```
        }
        break;
    }
    break;

// WORD
case 1:
    switch( EaMode )
    {
    case 0:     // Dn.w
        if ( decImm )sprintf ( buf, " addq.w #%d,d%X", Data, EaReg );
        else  sprintf ( buf, " addq.w #$%X,d%X", Data, EaReg );
        break;
    case 1:     // An.w
        if ( decImm )sprintf ( buf, " addq.w #%d,a%X", Data, EaReg );
        else  sprintf ( buf, " addq.w #$%X,a%X", Data, EaReg );
        break;
    case 2:     // (An).w
        if ( decImm )sprintf ( buf, " addq.w #%d,(a%X)", Data, EaReg );
        else  sprintf ( buf, " addq.w #$%X,(a%X)", Data, EaReg );
        break;
    case 3:     // (An)+.w
        if ( decImm )sprintf ( buf, " addq.w #%d,(a%X)+", Data, EaReg );
        else  sprintf ( buf, " addq.w #$%X,(a%X)+", Data, EaReg );
        break;
    case 4:     // -(An).w
        if ( decImm )sprintf ( buf, " addq.w #%d,-(a%X)", Data, EaReg );
        else  sprintf ( buf, " addq.w #$%X,-(a%X)", Data, EaReg );
        break;
    case 5:     // (d16, An).w
        Disp16 = memory->GetWord( _address + 2 );
        if ( decImm )sprintf ( buf, " addq.w #%d,%d(a%X)", Data, Disp16, EaReg );
        else  sprintf ( buf, " addq.w #$%X,$%X(a%X)", Data, Disp16, EaReg );

        _address += 2;
        break;
    case 6:     // Disp8(An, Xn).w
        Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
        Xn = GETBITS( _address + 2, 0x7000, 12 );
        Wl = GETBITS( _address + 2, 0x800, 11 );

        if ( !Wl )      // word index
            if ( decImm )sprintf ( buf, " addq.w #%d,%d(a%X,d%X.w)", Data, Disp8, EaReg, Xn );
            else  sprintf ( buf, " addq.w #$%X,$%X(a%X,d%X.w)", Data, Disp8, EaReg, Xn );
        else
            if ( decImm )sprintf ( buf, " addq.w #%d,%d(a%X,d%X.l)", Data, Disp8, EaReg, Xn );
            else  sprintf ( buf, " addq.w #$%X,$%X(a%X,d%X.l)", Data, Disp8, EaReg, Xn );

        _address += 2;
        break;
    case 7:
        switch( EaReg )
```

```
        {
            case 0:         // (www).W
                if( decImm )sprintf( buf, "addq.w #%d,$%X", Data, memory->GetWord( _address + 2 ));
                else  sprintf( buf, "addq.w #$%X,$%X", Data, memory->GetWord( _address + 2 ));

                _address += 2;
                break;
            case 1:         // (www).L
                if( decImm )sprintf( buf, "addq.l #%d,$%lX", Data, memory->GetLongword( _address + 2 ));
                else  sprintf( buf, "addq.l #$%X,$%lX", Data, memory->GetLongword( _address + 2 ));

                _address += 4;
                break;
            case 2:         // (d16, PC).w (n/a)
            case 3:         // d8(PC, Xn) (n/a)
            case 4:         // #<data>    (n/a)
            case 5:
            case 6:
            case 7:
                // illegal ();
                break;
        }
        break;
    }
    break;
// LONGWORD (Source: Dn)
case 2:
    switch( EaMode )
    {
    case 0:     // Dn.l
        if( decImm )sprintf( buf, "addq.l #%d,d%X", Data, EaReg );
        else  sprintf( buf, "addq.l #$%X,d%X", Data, EaReg );
        break;
    case 1:     // An.l (n/a for dest = Ea)
        if( decImm )sprintf( buf, "addq.l #%d,a%X", Data, EaReg );
        else  sprintf( buf, "addq.l #$%X,a%X", Data, EaReg );
        break;
    case 2:     // (An).l
        if( decImm )sprintf( buf, "addq.l #%d,(a%X)", Data, EaReg );
        else  sprintf( buf, "addq.l #$%X,(a%X)", Data, EaReg );
        break;
    case 3:     // (An)+.l
        if( decImm )sprintf( buf, "addq.l #%d,(a%X)+", Data, EaReg );
        else  sprintf( buf, "addq.l #$%X,(a%X)+", Data, EaReg );
        break;
    case 4:     // -(An).l
        if( decImm )sprintf( buf, "addq.l #%d,-(a%X)", Data, EaReg );
        else  sprintf( buf, "addq.l #$%X,-(a%X)", Data, EaReg );
        break;
    case 5:     // (d16, An).l
        Disp16 = memory->GetWord( _address + 2 );
        if( decImm )sprintf( buf, "addq.l #%d,%d(a%X)", Data, Disp16, EaReg );
```

```
            else sprintf ( buf, "addq.l #$%X,$%X(a%X)", Data, Disp16, EaReg );

            // Update PC
            _address += 2;
            break;
        case 6:     // Disp8(An, Xn).l
            Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
            Xn = GETBITS( _address + 2, 0x7000, 12 );
            Wl = GETBITS( _address + 2, 0x800, 11 );

            if ( !Wl )      // word index
                if ( decImm )sprintf ( buf, "addq.l #%d,%d(a%X,d%X.w)", Data, Disp8, EaReg, Xn );
                else sprintf ( buf, "addq.l #$%X,$%X(a%X,d%X.w)", Data, Disp8, EaReg, Xn );
            else
                if ( decImm )sprintf ( buf, "addq.l #%d,%d(a%X,d%X.l)", Data, Disp8, EaReg, Xn );
                else sprintf ( buf, "addq.l #$%X,$%X(a%X,d%X.l)", Data, Disp8, EaReg, Xn );

            // Update PC
            _address += 2;
            break;
        case 7:
            switch( EaReg )
            {
                case 0:         // (www).W
                    if ( decImm )sprintf ( buf, "addq.l #%d,$%X", Data, memory->GetWord( _address + 2 ));
                    else sprintf ( buf, "addq.l #$%X,$%X", Data, memory->GetWord( _address + 2 ));
                    _address += 2;
                    break;
                case 1:         // (www).L
                    if ( decImm )sprintf ( buf, "addq.l #%d,$%lX", Data, memory->GetLongword( _address + 2 ));
                    else sprintf ( buf, "addq.l #$%X,$%lX", Data, memory->GetLongword( _address + 2 ));
                    _address += 4;
                    break;
                case 2:         // (d16, PC).l (n/a)
                case 3:         // d8(PC, Xn) (n/a)
                case 4:         // #<data>    (n/a)
                case 5:
                case 6:
                case 7:
                    // illegal ();
                    break;
            }
            break;
        }
    }

    // Update _address
    _address += 2;

    return _address;
}
```

```
unsigned long M68008::tS_addx( unsigned long _address, char *buf )
{
    //
    // Alan Donnelly (15/04/02)
    //

    unsigned char Size = GET_TINSTRBITS(0xC0, 6);
    unsigned char Rx = GET_TINSTRBITS(0xE00, 9);
    unsigned char Ry = GET_TINSTRBITS(0x7, 0);
    unsigned char Rm = GET_TINSTRBITS(0x8, 3);

    // Check registers are correct (since this could be called from an illegal ADD)
    if( Rx > 7 || Ry > 7 ) illegal ();

    switch( Size )
    {
        case 0:     // BYTE
            if ( !Rm )  // Data Reg. to Data Reg.
                sprintf ( buf, "addx.b d%X,d%X", Ry, Rx );
            else        // Address to Address
                sprintf ( buf, "addx.b -(a%X),-(a%X)", Ry, Rx );
            break;
        case 1:     // WORD
            if ( !Rm )  // Data Reg. to Data Reg.
                sprintf ( buf, "addx.w d%X,d%X", Ry, Rx );
            else        // Address to Address
                sprintf ( buf, "addx.w -(a%X),-(a%X)", Ry, Rx );
            break;
        case 2:     // LONGWORD
            if ( !Rm )  // Data Reg. to Data Reg.
                sprintf ( buf, "addx.l d%X,d%X", Ry, Rx );
            else        // Address to Address
                sprintf ( buf, "addx.l -(a%X),-(a%X)", Ry, Rx );
            break;
        case 3:     // Probably ADDA
            return tS_add( _address, buf );
        default:
            illegal ();
    }

    _address += 2;

    return _address;
}

unsigned long M68008::tS_and( unsigned long _address, char *buf )
{
    //
    // Alan Donnelly (12/04/2002)
    //
```

```
//
// This also handles ADDA ( same instruction signature )
//
// check immediate and (xxx).W/L
//

unsigned char Reg = GETBITS(_address, 0xE00, 9);
unsigned char OpMode = GETBITS(_address, 0x1C0, 6);
unsigned char EaMode = GETBITS(_address, 0x38, 3);
unsigned char EaReg = GETBITS(_address, 0x7, 0);

// (d16, An)
signed short int Disp16;

// Disp8(An, Xn)
signed char Disp8;
unsigned char Xn;
unsigned char Wl;

// get addressing mode and source (Ea) value
// bool EaIsSource = GETBITS(pc, 0x100, 0) == 0;

switch( OpMode )
{
case 0:  // BYTE (Source: Ea)

    switch( EaMode )
    {
    case 0:      // Dn.b
        sprintf ( buf, "and.b␣d%X,d%X", EaReg, Reg);
        break;
    case 1:      // An.b (doesn't apply)
        tS_addx( _address, buf );
        break;
    case 2:      // (An).b
        sprintf ( buf, "and.b␣(a%X),d%X", EaReg, Reg);
        break;
    case 3:      // (An)+.b
        sprintf ( buf, "and.b␣(a%X)+,d%X", EaReg, Reg);
        break;
    case 4:      // -(An).b
        sprintf ( buf, "and.b␣-(a%X),d%X", EaReg, Reg );
        break;
    case 5:      // (d16, An).b
        Disp16 = memory->GetWord( _address + 2 );

        if ( decImm )sprintf ( buf, "and.b␣%d(a%X),d%X", Disp16, EaReg, Reg );
        else  sprintf ( buf, "and.b␣$%X(a%X),d%X", Disp16, EaReg, Reg );

        // modify _address
        _address += 2;
```

```
    break;
case 6:      // Disp8(An, Xn).b
    // +---+---+---+---+-----+---+---+---+---+---+---+---+---+---+---+---+
    // | 0 |    Xn   | w/l  | 0 | 0 | 0 |          Disp8                |
    // +---+---+---+---+-----+---+---+---+---+---+---+---+---+---+---+---+
    //

    Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
    Xn = GETBITS( _address + 2, 0x7000, 12 );
    Wl = GETBITS( _address + 2, 0x800, 11 );

    if ( !Wl )       // word index
        if ( decImm )sprintf( buf, "and.b %d(a%X,d%X.w),d%X", Disp8, EaReg, Xn, Reg );
        else sprintf( buf, "and.b $%X(a%X,d%X.w),d%X", Disp8, EaReg, Xn, Reg );
    else
        if ( decImm )sprintf( buf, "and.b %d(a%X,d%X.l),d%X", Disp8, EaReg, Xn, Reg );
        else sprintf( buf, "and.b $%X(a%X,d%X.l),d%X", Disp8, EaReg, Xn, Reg );

    _address += 2;    // modify _address

    break;
case 7:
    switch( EaReg )
    {
        case 0:          // (www).W
            sprintf( buf, "and.b $%X,d%X", memory->GetWord( _address + 2 ), Reg );

            _address += 2; // modify _address
            break;
        case 1:          // (www).L
            sprintf( buf, "and.b $%lX,d%X", memory->GetLongword( _address + 2 ), Reg );

            _address += 4; // modify _address
            break;
        case 2:          // (d16, PC).b
            Disp16 = memory->GetWord( _address + 2 );
            if ( decImm )sprintf( buf, "and.b %ld(PC),d%X", Disp16 + 2 + _address, Reg );
            else sprintf( buf, "and.b $%lX(PC),d%X", Disp16 + 2 + _address, Reg );

            _address += 2; // modify _address
            break;
        case 3:          // d8(PC, Xn)
            Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
            Xn = GETBITS( _address + 2, 0x7000, 12 );
            Wl = GETBITS( _address + 2, 0x800, 11 );

            if ( !Wl )   // word index
                if ( decImm )sprintf( buf, "and.b %ld(PC,d%X.w),d%X", Disp8 + 2 + _address, Xn, Reg );
                else sprintf( buf, "and.b $%lX(PC,d%X.w),d%X", Disp8 + 2 + _address, Xn, Reg );
            else      // long index
                if ( decImm )sprintf( buf, "and.b %ld(PC,d%X.l),d%X", Disp8 + 2 + _address, Xn, Reg );
                else sprintf( buf, "and.b $%lX(PC,d%X.l),d%X", Disp8 + 2 + _address, Xn, Reg );
```

```
            _address += 2; // modify _address
            break;
        case 4:          // #<data>
            if( decImm )sprintf( buf, "and.b #%d,d%X", memory->GetByte( _address + 3 ), Reg );
            else sprintf( buf, "and.b #$%X,d%X", memory->GetByte( _address + 3 ), Reg );
            _address += 2;    // modify _address
            break;
        case 5:
        case 6:
        case 7:
            tS_addx( _address, buf );
            break;
      }
      break;
  }
break;

case 1:
  switch( EaMode )
  {
  // WORD (Source: Ea)
  case 0:      // Dn.w
      sprintf( buf, "and.w d%X,d%X", EaReg, Reg );
      break;
  case 1:      // An.w
      sprintf( buf, "and.w a%X,d%X", EaReg, Reg );
      break;
  case 2:      // (An).w
      sprintf( buf, "and.w (a%X),d%X", EaReg, Reg );
      break;
  case 3:      // (An)+.w
      sprintf( buf, "and.w (a%X)+,d%X", EaReg, Reg );
      break;
  case 4:      // -(An).w
      sprintf( buf, "and.w -(a%X),d%X", EaReg, Reg );
      break;
  case 5:  // (d16, An).w
      Disp16 = memory->GetWord( _address + 2 );
      if( decImm )sprintf( buf, "and.w %d(a%X),d%X", Disp16, EaReg, Reg );
      else sprintf( buf, "and.w $%X(a%X),d%X", Disp16, EaReg, Reg );

      _address += 2;    // modify _address
      break;
  case 6:      // Disp8(An, Xn).w
      Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
      Xn = GETBITS( _address + 2, 0x7000, 12 );
      Wl = GETBITS( _address + 2, 0x800, 11 );

      if ( !Wl )       // word index
          if( decImm )sprintf( buf, "and.w %d(a%X,d%X.w),d%X", Disp8, EaReg, Xn, Reg );
          else sprintf( buf, "and.w $%X(a%X,d%X.w),d%X", Disp8, EaReg, Xn, Reg );
```

```
    else
        if ( decImm )sprintf( buf, " and.w␣%d(a%X,d%X.l),d%X", Disp8, EaReg, Xn, Reg );
        else  sprintf ( buf, " and.w␣$%X(a%X,d%X.l),d%X", Disp8, EaReg, Xn, Reg );

    _address += 2;     // modify _address
    break;
case 7:
    switch( EaReg )
    {
        case 0:          // (www).W
            sprintf ( buf, " and.w␣$%X,d%X", memory->GetWord( _address + 2 ), Reg );

            _address += 2; // modify _address
            break;
        case 1:          // (www).L
            sprintf ( buf, " and.w␣$%lX,d%X", memory->GetLongword( _address + 2 ), Reg );

            _address += 4; // modify _address
            break;
        case 2:          // (d16, PC).w
            Disp16 = memory->GetWord( _address + 2 );
            if ( decImm )sprintf ( buf, " and.w␣%ld(PC),d%X", Disp16 + 2 + _address, Reg );
            else  sprintf ( buf, " and.w␣$%lX(PC),d%X", Disp16 + 2 + _address, Reg );

            _address += 2; // modify _address
            break;
        case 3:          // d8(PC, Xn)
            Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
            Xn = GETBITS( _address + 2, 0x7000, 12 );
            Wl = GETBITS( _address + 2, 0x800, 11 );

            if ( !Wl )    // word index
                if ( decImm )sprintf ( buf, " and.w␣%ld(PC,d%X.w),d%X", Disp8 + 2 + _address, Xn, Reg );
                else  sprintf ( buf, " and.w␣$%lX(PC,d%X.w),d%X", Disp8 + 2 + _address, Xn, Reg );
            else       // long index
                if ( decImm )sprintf ( buf, " and.w␣%ld(PC,d%X.l),d%X", Disp8 + 2 + _address, Xn, Reg );
                else  sprintf ( buf, " and.w␣$%lX(PC,d%X.l),d%X", Disp8 + 2 + _address, Xn, Reg );

            _address += 2; // modify _address
            break;
        case 4:          // #<data>
            if ( decImm )sprintf ( buf, " and.w␣#%d,d%X", memory->GetWord( _address + 2), Reg );
            else  sprintf ( buf, " and.w␣#$%X,d%X", memory->GetWord( _address + 2), Reg );

            _address += 2; // modify _address
            break;
        case 5:
        case 6:
        case 7:
            tS_addx( _address, buf );
            break;
    }
```

```
        break;
    }

    break;

case 2:     // LONGWORD (Source: Ea)

    switch( EaMode )
    {
    case 0:     // Dn.l
        sprintf ( buf, "and.l␣d%X,d%X", EaReg, Reg );
        break;
    case 1:     // An.l
        sprintf ( buf, "and.l␣a%X,d%X", EaReg, Reg );
        break;
    case 2:     // (An).l
        sprintf ( buf, "and.l␣(a%X),d%X", EaReg, Reg );
        break;
    case 3:     // (An)+.l
        sprintf ( buf, "and.l␣(a%X)+,d%X", EaReg, Reg );
        break;
    case 4:     // -(An).l
        sprintf ( buf, "and.l␣-(a%X),d%X", EaReg, Reg );
        break;
    case 5:     // (d16, An).l
        Disp16 = memory->GetWord( _address + 2 );
        if ( decImm )sprintf ( buf, "and.l␣%d(a%X),d%X", Disp16, EaReg, Reg );
        else  sprintf ( buf, "and.l␣$%X(a%X),d%X", Disp16, EaReg, Reg );

        _address += 2; // modify _address
        break;
    case 6:     // Disp8(An, Xn).l
        Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
        Xn = GETBITS( _address + 2, 0x7000, 12 );
        Wl = GETBITS( _address + 2, 0x800, 11 );

        if ( !Wl )   // word index
            if ( decImm )sprintf ( buf, "and.l␣%d(a%X,d%X.w),d%X", Disp8, EaReg, Xn, Reg );
            else  sprintf ( buf, "and.l␣$%X(a%X,d%X.w),d%X", Disp8, EaReg, Xn, Reg );
        else     // long index
            if ( decImm )sprintf ( buf, "and.l␣%d(a%X,d%X.l),d%X", Disp8, EaReg, Xn, Reg );
            else  sprintf ( buf, "and.l␣$%X(a%X,d%X.l),d%X", Disp8, EaReg, Xn, Reg );

        _address += 2; // modify _address
        break;
    case 7:
        switch( EaReg )
        {
            case 0:          // (www).W
                sprintf ( buf, "and.l␣$%X,d%X", memory->GetWord( _address + 2 ), Reg );

                _address += 2; // modify _address
```

```
            break;
        case 1:         // (www).L
            sprintf ( buf, " and.l␣$%X,d%X", memory->GetWord( _address + 2 ), Reg );

            _address += 4; // modify _address
            break;
        case 2:         // (d16, PC).l
            Disp16 = memory->GetWord( _address + 2 );
            if ( decImm )sprintf ( buf, " and.l␣%ld(PC),d%X", Disp16 + 2 + _address, Reg );
            else  sprintf ( buf, " and.l␣$%lX(PC),d%X", Disp16 + 2 + _address, Reg );

            _address += 2; // modify _address
            break;
        case 3:         // d8(PC, Xn)
            Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
            Xn = GETBITS( _address + 2, 0x7000, 12 );
            Wl = GETBITS( _address + 2, 0x800, 11 );

            if ( !Wl )   // word index
                if ( decImm )sprintf ( buf, " and.l␣%ld(PC,d%X.w),d%X", Disp8 + 2 + _address, Xn, Reg );
                else  sprintf ( buf, " and.l␣$%lX(PC,d%X.w),d%X", Disp8 + 2 + _address, Xn, Reg );
            else      // long index
                if ( decImm )sprintf ( buf, " and.l␣%ld(PC,d%X.l),d%X", Disp8 + 2 + _address, Xn, Reg );
                else  sprintf ( buf, " and.l␣$%lX(PC,d%X.l),d%X", Disp8 + 2 + _address, Xn, Reg );

            _address += 2; // modify _address
            break;
        case 4:         // #<data>
            if ( decImm )sprintf ( buf, " and.l␣#%ld,d%X", memory->GetLongword( _address + 2 ), Reg );
            else  sprintf ( buf, " and.l␣#$%lX,d%X", memory->GetLongword( _address + 2 ), Reg );

            _address += 4; // modify _address
            break;
        case 5:
        case 6:
        case 7:
            tS_addx( _address, buf );
            break;
    }

  break;
  }
break;

case 4:     // BYTE (Source: Dn)

  switch( EaMode )
  {
  case 0:     // Dn.b (n/a for dest = Ea)
  case 1:     // An.b (n/a for dest = Ea)
    tS_addx( _address, buf );
    break;
```

```
case 2:     // (An).b
    sprintf ( buf, "and.b␣d%X,(a%X)", Reg, EaReg );
    break;
case 3:     // (An)+.b
    sprintf ( buf, "and.b␣d%X,(a%X)+", Reg, EaReg );
    break;
case 4:     // -(An).b
    sprintf ( buf, "and.b␣d%X,-(a%X)", Reg, EaReg );
    break;
case 5:     // (d16, An).b
    Disp16 = memory->GetWord( _address + 2 );

    if ( decImm )sprintf ( buf, "and.b␣d%X,%d(a%X)", Reg, Disp16, EaReg );
    else  sprintf ( buf, "and.b␣d%X,$%X(a%X)", Reg, Disp16, EaReg );

    _address += 2; // modify _address
    break;
case 6:     // Disp8(An, Xn).b
    Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
    Xn = GETBITS( _address + 2, 0x7000, 12 );
    Wl = GETBITS( _address + 2, 0x800, 11 );

    if ( !Wl )   // word index
        if ( decImm )sprintf ( buf, "and.b␣d%X,%d(a%X,d%X.w)", Reg, Disp8, EaReg, Xn );
        else  sprintf ( buf, "and.b␣d%X,$%X(a%X,d%X.w)", Reg, Disp8, EaReg, Xn );
    else     // long index
        if ( decImm )sprintf ( buf, "and.b␣d%X,%d(a%X,d%X.l)", Reg, Disp8, EaReg, Xn );
        else  sprintf ( buf, "and.b␣d%X,$%X(a%X,d%X.l)", Reg, Disp8, EaReg, Xn );

    _address += 2; // modify _address
    break;
case 7:
    switch( EaReg )
    {
        case 0:         // (www).W
            sprintf ( buf, "and.b␣d%X,$%X", Reg, memory->GetWord( _address + 2 ));

            _address += 2; // modify _address
            break;
        case 1:         // (www).L
            sprintf ( buf, "and.b␣d%X,$%lX", Reg, memory->GetLongword( _address + 2 ));

            _address += 4; // modify _address
            break;
        case 2:         // (d16, PC).b
        case 3:         // d8(PC, Xn)
        case 4:         // #<data>
        case 5:
        case 6:
        case 7:
            tS_addx( _address, buf );
            break;
```

```
        }
        break;
    }
    break;

case 5:     // WORD (Source: Dn)
    switch( EaMode )
    {
    case 0:     // Dn.w (n/a for dest = Ea)
    case 1:     // An.w (n/a for dest = Ea)
        tS_addx( _address, buf );
        break;
    case 2:     // (An).w
        sprintf( buf, "and.w d%X,(a%X)", Reg, EaReg );
        break;
    case 3:     // (An)+.w
        sprintf( buf, "and.w d%X,(a%X)+", Reg, EaReg );
        break;
    case 4:     // -(An).w
        sprintf( buf, "and.w d%X,-(a%X)", Reg, EaReg );
        break;
    case 5:     // (d16, An).w
        Disp16 = memory->GetWord( _address + 2 );
        if( decImm )sprintf( buf, "and.w d%X,%d(a%X)", Reg, Disp16, EaReg );
        else sprintf( buf, "and.w d%X,$%X(a%X)", Reg, Disp16, EaReg );

        _address += 2; // modify _address
        break;
    case 6:     // Disp8(An, Xn).w
        Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
        Xn = GETBITS( _address + 2, 0x7000, 12 );
        Wl = GETBITS( _address + 2, 0x800, 11 );

        if( !Wl )   // word index
            if( decImm )sprintf( buf, "and.w d%X,%d(a%X,d%X.w)", Reg, Disp8, EaReg, Xn );
            else sprintf( buf, "and.w d%X,$%X(a%X,d%X.w)", Reg, Disp8, EaReg, Xn );
        else        // long index
            if( decImm )sprintf( buf, "and.w d%X,%d(a%X,d%X.l)", Reg, Disp8, EaReg, Xn );
            else sprintf( buf, "and.w d%X,$%X(a%X,d%X.l)", Reg, Disp8, EaReg, Xn );

        _address += 2; // modify _address
        break;
    case 7:
        switch( EaReg )
        {
            case 0:         // (www).W
                sprintf( buf, "and.w d%X,$%X", Reg, memory->GetWord( _address + 2) );

                _address += 2; // modify _address
                break;
            case 1:         // (www).L
                sprintf( buf, "and.w d%X,$%lX", Reg, memory->GetLongword( _address + 2) );
```

```
            _address += 4; // modify _address
            break;
        case 2:         // (d16, PC).w (n/a for dest = Ea)
        case 3:         // d8(PC, Xn) (n/a for dest = Ea)
        case 4:         // #<data>   (n/a for dest = Ea)
        case 5:
        case 6:
        case 7:
            tS_addx( _address, buf );
            break;
    }
    break;
}

    break;

case 6:     // LONGWORD (Source: Dn)
    switch( EaMode )
    {
        case 0:     // Dn.l (n/a for dest = Ea)
        case 1:     // An.l (n/a for dest = Ea)
            tS_addx( _address, buf );
            break;
        case 2:     // (An).l
            sprintf( buf, "and.l␣d%X,(a%X)", Reg, EaReg );
            break;
        case 3:     // (An)+.l
            sprintf( buf, "and.l␣d%X,(a%X)+", Reg, EaReg );
            break;
        case 4:     // -(An).l
            sprintf( buf, "and.l␣d%X,-(a%X)", Reg, EaReg );
            break;
        case 5:     // (d16, An).l
            Disp16 = memory->GetWord( _address + 2);
            if( decImm )sprintf( buf, "and.l␣d%X,%d(a%X)", Reg, Disp16, EaReg );
            else  sprintf( buf, "and.l␣d%X,$%X(a%X)", Reg, Disp16, EaReg );

            _address += 2; // modify _address
            break;
        case 6:     // Disp8(An, Xn).l
            Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
            Xn = GETBITS( _address + 2, 0x7000, 12 );
            Wl = GETBITS( _address + 2, 0x800, 11 );

            if ( !Wl )   // word index
                if( decImm )sprintf( buf, "and.l␣d%X,%d(a%X,d%X.w)", Reg, Disp8, EaReg, Xn );
                else  sprintf( buf, "and.l␣d%X,$%X(a%X,d%X.w)", Reg, Disp8, EaReg, Xn );
            else     // long index
                if( decImm )sprintf( buf, "and.l␣d%X,%d(a%X,d%X.l)", Reg, Disp8, EaReg, Xn );
                else  sprintf( buf, "and.l␣d%X,$%X(a%X,d%X.l)", Reg, Disp8, EaReg, Xn );
```

```
            _address += 2; // modify _address
            break;
        case 7:
            switch( EaReg )
            {
                case 0:         // (www).W
                    sprintf ( buf, " and.l_d%X,$%X", Reg, memory->GetWord( _address + 2 ));

                    _address += 2; // modify _address
                    break;
                case 1:         // (www).L
                    sprintf ( buf, " and.l_d%X,$%lX", Reg, memory->GetLongword( _address + 2 ));

                    _address += 4; // modify _address
                    break;
                case 2:         // (d16, PC).l (n/a for dest = Ea)
                case 3:         // d8(PC, Xn) (n/a for dest = Ea)
                case 4:         // #<data>    (n/a for dest = Ea)
                case 5:
                case 6:
                case 7:
                    tS_addx( _address, buf );
                    break;
            }
            break;
        }

        break;
    case 3:
        illegal ();
        break;
    case 7:
        illegal ();
        break;
    }

    // Update the _address
    _address += 2;

    return _address;
}

unsigned long M68008::tS_andi( unsigned long _address, char *buf )
{
    //
    // Alan Donnelly (13/04/02)
    //

    unsigned char Size = GET_TINSTRBITS(0xC0, 6);
    unsigned char EaMode = GET_TINSTRBITS(0x38, 3);
    unsigned char EaReg = GET_TINSTRBITS(0x7, 0);
```

```
// (d16, An)
signed short int Disp16;

// Disp8(An, Xn)
signed char Disp8;
unsigned char Xn;
unsigned char Wl;

DATA_REGISTER( Data );

switch( Size )
{
   case 0:      // BYTE

      // Uses the LSByte of the word following the  instruction
      Data.b = memory->GetWord( _address + 2 )& 0xFF;
      _address += 2;

      switch( EaMode )
      {
         case 0:      // Dn.b
            if ( decImm )sprintf ( buf, "andi.b #%d,d%X", Data.b, EaReg );
            else  sprintf ( buf, "andi.b #$%X,d%X", Data.b, EaReg );
            break;
         case 1:      // An.b (n/a)
             illegal ();
            break;
         case 2:      // (An).b
            if ( decImm )sprintf ( buf, "andi.b #%d,(a%X)", Data.b, EaReg );
            else  sprintf ( buf, "andi.b #$%X,(a%X)", Data.b, EaReg );
            break;
         case 3:      // (An)+.b
            if ( decImm )sprintf ( buf, "andi.b #%d,(a%X)+", Data.b, EaReg );
            else  sprintf ( buf, "andi.b #$%X,(a%X)+", Data.b, EaReg );
            break;
         case 4:      // --(An).b
            if ( decImm )sprintf ( buf, "andi.b #%d,-(a%X)", Data.b, EaReg );
            else  sprintf ( buf, "andi.b #$%X,-(a%X)", Data.b, EaReg );
            break;
         case 5:      // Disp16(An).b
            Disp16 = memory->GetWord( _address + 2 );
            if ( decImm )sprintf ( buf, "andi.b #%d,%d(a%X)", Data.b, Disp16, EaReg );
            else  sprintf ( buf, "andi.b #$%X,$%X(a%X)", Data.b, Disp16, EaReg );

            _address += 2;
            break;
         case 6:      // Disp8(An, Xn).b
            Disp8 = memory->GetByte( _address + 3 );
            Xn = GETBITS( _address + 2, 0x7000, 12 );
            Wl = GETBITS( _address + 2, 0x800, 11 );

            if  ( ! Wl )
```

```
            if( decImm )sprintf( buf, "andi.b #%d,%d(a%X,d%X.w)", Data.b, Disp8, EaReg, Xn );
            else sprintf( buf, "andi.b #$%X,$%X(a%X,d%X.w)", Data.b, Disp8, EaReg, Xn );
         else
            if( decImm )sprintf( buf, "andi.b #%d,%d(a%X,d%X.l)", Data.b, Disp8, EaReg, Xn );
            else sprintf( buf, "andi.b #$%X,$%X(a%X,d%X.l)", Data.b, Disp8, EaReg, Xn );

         _address += 2;
         break;
      case 7:
         switch ( EaReg )
         {
            case 0: // (xxx).W.b
               if( decImm )sprintf( buf, "andi.b #%d,$%X", Data.b, memory->GetWord( _address
                     + 2 ));
               else sprintf( buf, "andi.b #$%X,$%X", Data.b, memory->GetWord( _address + 2 ));
               _address += 2;
               break;
            case 1: // (xxx).L.b
               if( decImm )sprintf( buf, "andi.b #%d,$%lX", Data.b, memory->GetLongword(
                     _address + 2 ));
               else sprintf( buf, "andi.b #$%X,$%lX", Data.b, memory->GetLongword( _address + 2 )
                     );
               _address += 4;
               break;
            default:
               illegal ();
               break;
         }
         break;
   }

   break;

case 1:     // WORD
   Data.w = memory->GetWord( _address + 2 );
   _address += 2;

   switch( EaMode )
   {
      case 0:     // Dn.w
         if( decImm )sprintf( buf, "andi.w #%d,d%X", Data.w, EaReg );
         else sprintf( buf, "andi.w #$%X,d%X", Data.w, EaReg );
         break;
      case 1:     // An.w
         if( decImm )sprintf( buf, "andi.w #%d,a%X", Data.w, EaReg );
         else sprintf( buf, "andi.w #$%X,a%X", Data.w, EaReg );
         break;
      case 2:     // (An).w
         if( decImm )sprintf( buf, "andi.w #%d,(a%X)", Data.w, EaReg );
         else sprintf( buf, "andi.w #$%X,(a%X)", Data.w, EaReg );
         break;
      case 3:     // (An)+.w
```

```
            if ( decImm )sprintf( buf, "andi.w #%d,(a%X)+", Data.w, EaReg );
            else sprintf( buf, "andi.w #$%X,(a%X)+", Data.w, EaReg );
            break;
        case 4:     // -(An).w
            if ( decImm )sprintf( buf, "andi.w #%d,-(a%X)", Data.w, EaReg );
            else sprintf( buf, "andi.w #$%X,-(a%X)", Data.w, EaReg );
            break;
        case 5:     // Disp16(An).w
            Disp16 = memory->GetWord( _address + 2 );
            if ( decImm )sprintf( buf, "andi.w #%d,%d(a%X)", Data.w, Disp16, EaReg );
            else sprintf( buf, "andi.w #$%X,$%X(a%X)", Data.w, Disp16, EaReg );

            _address += 2;
            break;
        case 6:     // Disp8(An, Xn).w
            Disp8 = memory->GetByte( _address + 3 );
            Xn = GETBITS( _address + 2, 0x7000, 12 );
            Wl = GETBITS( _address + 2, 0x800, 11 );
            if ( !Wl )
                if ( decImm )sprintf( buf, "andi.w #%d,%d(a%X,d%X.w)", Data.w, Disp8, EaReg, Xn );
                else sprintf( buf, "andi.w #$%X,$%X(a%X,d%X.w)", Data.w, Disp8, EaReg, Xn );
            else
                if ( decImm )sprintf( buf, "andi.w #%d,%d(a%X,d%X.l)", Data.w, Disp8, EaReg, Xn );
                else sprintf( buf, "andi.w #$%X,$%X(a%X,d%X.l)", Data.w, Disp8, EaReg, Xn );

            _address += 2;
            break;
        case 7:
            switch ( EaReg )
            {
                case 0: // (xxx).W.w
                    if ( decImm )sprintf( buf, "andi.w #%d,$%X", Data.w, memory->GetWord( _address
                        + 2 ));
                    else sprintf( buf, "andi.w #$%X,$%X", Data.w, memory->GetWord( _address + 2 ));

                    _address += 2;
                    break;
                case 1: // (xxx).L.w
                    if ( decImm )sprintf( buf, "andi.w #%d,$%lX", Data.w, memory->GetLongword(
                        _address + 2 ));
                    else sprintf( buf, "andi.w #$%X,$%lX", Data.w, memory->GetLongword( _address
                        + 2 ));

                    _address += 4;
                    break;
                default:
                    illegal ();
                    break;
            }
            break;
    }
    break;
```

```
case 2: // LONGWORD
Data.l = memory->GetLongword( _address + 2 );
_address += 4;

switch( EaMode )
{
   case 0:      // Dn.l
      if( decImm )sprintf( buf, "andi.l #%ld,d%X", Data.l, EaReg );
      else sprintf( buf, "andi.l #$%lX,d%X", Data.l, EaReg );
      break;
   case 1:      // An.l
      if( decImm )sprintf( buf, "andi.l #%ld,a%X", Data.l, EaReg );
      else sprintf( buf, "andi.l #$%lX,a%X", Data.l, EaReg );
      break;
   case 2:      // (An).l
      if( decImm )sprintf( buf, "andi.l #%ld,(a%X)", Data.l, EaReg );
      else sprintf( buf, "andi.l #$%lX,(a%X)", Data.l, EaReg );
      break;
   case 3:      // (An)+.l
      if( decImm )sprintf( buf, "andi.l #%ld,(a%X)+", Data.l, EaReg );
      else sprintf( buf, "andi.l #$%lX,(a%X)+", Data.l, EaReg );
      break;
   case 4:      // --(An).l
      if( decImm )sprintf( buf, "andi.l #%ld,-(a%X)", Data.l, EaReg );
      else sprintf( buf, "andi.l #$%lX,-(a%X)", Data.l, EaReg );
      break;
   case 5:      // Disp16(An).l
      Disp16 = memory->GetWord( _address + 2 );
      if( decImm )sprintf( buf, "andi.l #%ld,%d(a%X)", Data.l, Disp16, EaReg );
      else sprintf( buf, "andi.l #$%lX,$%X(a%X)", Data.l, Disp16, EaReg );

      _address += 2;
      break;
   case 6:      // Disp8(An, Xn).l
      Disp8 = memory->GetByte( _address + 3 );
      Xn = GETBITS( _address + 2, 0x7000, 12 );
      Wl = GETBITS( _address + 2, 0x800, 11 );
      if ( !Wl )
         if( decImm )sprintf( buf, "andi.l #%ld,%d(a%X,d%X.w)", Data.l, Disp8, EaReg, Xn );
         else sprintf( buf, "andi.l #$%lX,$%X(a%X,d%X.w)", Data.l, Disp8, EaReg, Xn );
      else
         if( decImm )sprintf( buf, "andi.l #%ld,%d(a%X,d%X.l)", Data.l, Disp8, EaReg, Xn );
         else sprintf( buf, "andi.l #$%lX,$%X(a%X,d%X.l)", Data.l, Disp8, EaReg, Xn );

      _address += 2;
      break;
   case 7:
      switch ( EaReg )
      {
         case 0: // (xxx).W.l
```

```
                    if ( decImm )sprintf( buf, "andi.l #%ld,$%X", Data.l, memory->GetWord( _address
                        + 2) );
                    else sprintf( buf, "andi.l #$%lX,$%X", Data.l, memory->GetWord( _address + 2) );

                    _address += 2;
                    break;
                case 1: // (xxx).L.l
                    if ( decImm )sprintf( buf, "andi.l #%ld,$%lX", Data.l, memory->GetLongword(
                        _address + 2) );
                    else sprintf( buf, "andi.l #$%lX,$%lX", Data.l, memory->GetLongword( _address + 2)
                        );

                    _address += 4;
                    break;
                default:
                    illegal ();
                    break;
            }
            break;
        }
        break;
    }

    // Update _address
    _address += 2;

    return _address;
}

unsigned long M68008::tS_andiCcr( unsigned long _address, char *buf )
{
    //
    // Alan Donnelly (15/04/02)
    //

    unsigned char Data = memory->GetByte( _address + 3 );

    if ( decImm )sprintf( buf, "andi.b #%d,CCR", Data );
    else sprintf( buf, "andi.b #$%X,CCR", Data );

    _address += 4;


    return _address;
}

unsigned long M68008::tS_asl_rMem( unsigned long _address, char *buf )
{
    //
    // Alan Donnelly (15/04/02)
    // (Note: ASL/R on memory operands only shift once per instruction
    //         and shifts can only be performed on words)
```

```
//

bool left = GET_TINSTRBITS( 0x100, 8 )== 1;  // Direction
unsigned char EaMode = GET_TINSTRBITS(0x38, 3); // Addressing mode
unsigned char EaReg = GET_TINSTRBITS(0x7, 0); // Register

// (d16, An)
signed short int Disp16;

// Disp8(An, Xn)
signed char Disp8;
unsigned char Xn;
unsigned char Wl;

switch ( EaMode )
{
   case 0: // Dn (n/a)
   case 1: // An (n/a)
       illegal ();
       break;
   case 2: // (An)
       if( left )
           sprintf( buf, " asl.w (a%X)", EaReg );
       else
           sprintf( buf, " asr.w (a%X)", EaReg );

       break;
   case 3: // (An)+
       if( left )
           sprintf( buf, " asl.w (a%X)+", EaReg );
       else
           sprintf( buf, " asr.w (a%X)+", EaReg );

       break;
   case 4: // -(An)
       if( left )
           sprintf( buf, " asl.w -(a%X)", EaReg );
       else
           sprintf( buf, " asr.w -(a%X)", EaReg );

       break;
   case 5: // Disp16(An)
       Disp16 = memory->GetWord( _address + 2);

       if( left )
           if( decImm )sprintf( buf, " asl.w %d(a%X)", Disp16, EaReg );
           else  sprintf( buf, " asl.w $%X(a%X)", Disp16, EaReg );
       else
           if( decImm )sprintf( buf, " asr.w %d(a%X)", Disp16, EaReg );
           else  sprintf( buf, " asr.w $%X(a%X)", Disp16, EaReg );

       // Update PC
```

```
    _address += 2;

    break;
case 6:  // Disp8(An, Xn)
    Disp8 = memory->GetByte( _address + 3 );
    Xn = GETBITS( _address + 2, 0x7000, 12 );
    Wl = GETBITS( _address + 2, 0x800, 11 );

    if ( !Wl )
    {
        if ( left )
            if ( decImm )sprintf( buf, " asl.w %d(a%X,d%X.w)", Disp8, EaReg, Xn );
            else sprintf( buf, " asl.w $%X(a%X,d%X.w)", Disp8, EaReg, Xn );
        else
            if ( decImm )sprintf( buf, " asr.w %d(a%X,d%X.w)", Disp8, EaReg, Xn );
            else sprintf( buf, " asr.w $%X(a%X,d%X.w)", Disp8, EaReg, Xn );
    }
    else
    {
        if ( left )
            if ( decImm )sprintf( buf, " asl.w %d(a%X,d%X.l)", Disp8, EaReg, Xn );
            else sprintf( buf, " asl.w $%X(a%X,d%X.l)", Disp8, EaReg, Xn );
        else
            if ( decImm )sprintf( buf, " asr.w %d(a%X,d%X.l)", Disp8, EaReg, Xn );
            else sprintf( buf, " asr.w $%X(a%X,d%X.l)", Disp8, EaReg, Xn );
    }

    // Update PC
    _address += 2;

    break;

case 7:
    switch( EaReg )
    {
        case 0:  // (xxx).W

            if ( left )
                sprintf( buf, " asl.w $%X", memory->GetWord( _address + 2 ));
            else
                sprintf( buf, " asr.w $%X", memory->GetWord( _address + 2 ));

            // Update PC
            _address += 2;

            break;
        case 1:  // (xxx).L

            if ( left )
                sprintf( buf, " asl.w $%lX", memory->GetLongword( _address + 2 ));
            else
                sprintf( buf, " asr.w $%lX", memory->GetLongword( _address + 2 ));
```

```
                // Update PC
                _address += 4;

                break;
            default:
                illegal();
                break;
        }
    }

    // Update PC
    _address += 2;

    return _address;
}

unsigned long M68008::tS_asl_rReg( unsigned long _address, char *buf )
{
    //
    // Alan Donnelly (14/04/02)
    //     Register shifts
    //

    unsigned char Size = GET_TINSTRBITS(0xC0, 6);  // Size: b, w, l
    unsigned char Reg = GET_TINSTRBITS(0x7, 0);   // Register to shift
    bool left = GET_TINSTRBITS( 0x100, 8 )== 1;  // Shift direction
    unsigned char ImmReg = GET_TINSTRBITS(0xE00, 9); // Either an immediate value for the amount
                                                     // to shift by or the number of the data
                                                     // register holding the shift count
    char *RegImm;                                    // char is 'd' for regs. '#' for imm.

    if ( GET_TINSTRBITS( 0x20, 5 ))                  // Register contains shift amount
        RegImm = "d";                                // mod 64 according to Motorola manual (pp. 127)
    else                                             // Immediate shift value
    {
        if( ImmReg == 0 )ImmReg = 8;

        if( decImm )
            RegImm = "#";
        else
            RegImm = "#$";
    }

    switch ( Size )
    {
        case 0:          // BYTE
            if ( left ) // Left
                sprintf( buf, " asl.b_%s%X,d%d", RegImm, ImmReg, Reg );
            else       // Right
                sprintf( buf, " asr.b_%s%X,d%d", RegImm, ImmReg, Reg );
            break;
```

```
        case 1:          // WORD
            if ( left ) // Left
                sprintf( buf, " asl.w %s%X,d%d", RegImm, ImmReg, Reg );
            else       // Right
                sprintf( buf, " asr.w %s%X,d%d", RegImm, ImmReg, Reg );
            break;
        case 2:          // LONGWORD
            if ( left ) // Left
                sprintf( buf, " asl.l %s%X,d%d", RegImm, ImmReg, Reg );
            else       // Right
                sprintf( buf, " asr.l %s%X,d%d", RegImm, ImmReg, Reg );
            break;
    }

    // Update the _address
    _address += 2;

    return _address;
}

unsigned long M68008::tS_bxx( unsigned long _address, char *buf )
{
    //
    // Gerard Whyte (16/04/02)
    //

    unsigned char Condition = GET_TINSTRBITS(0xF00, 8);
    signed char Disp8 = GET_TINSTRBITS(0xFF, 0);
    signed short int Disp16 = 0;
    char size [ 32 ];

    if( Disp8 == -1 )  // 32-bit disp not supported
    {
        sprintf( buf, "32-Bit Displacement not supported for bcc" );
        _address += 6;
    }
    else
    {
        if ( Disp8 == 0 ) // 16-bit imm. disp if 8-bit disp == 0
        {
            Disp16 = memory->GetWord( _address + 2 );
            sprintf( size, ".w $%lX", Disp16 + _address + 2);
            _address += 4;
        }
        else                // 8-bit displacement
        {
            sprintf( size, ".b $%lX", Disp8 + _address + 2);
            _address += 2;
        }

        switch( Condition )
        {
```

```
            case 0:     // True  (not defined)
            case 1:     // False (not defined)
                sprintf ( buf, " Not Defined! Illegal ! Get a better assembler!");
            case 2:     // BHI (HI = !C & !Z)
                sprintf ( buf, " bhi%s", size);
                break;
            case 3:     // BLS (LS = C | V)
                sprintf ( buf, " bls%s", size);
                break;
            case 4:     // BCC (CC = !C)
                sprintf ( buf, " bcc%s", size);
                break;
            case 5:     // BCS (CS = C)
                sprintf ( buf, " bcs%s", size);
                break;
            case 6:     // BNE (NE = !Z)
                sprintf ( buf, " bne%s", size);
                break;
            case 7:     // BEQ (EQ = Z)
                sprintf ( buf, " beq%s", size);
                break;
            case 8:     // BVC (VC = !V)
                sprintf ( buf, " bvc%s", size);
                break;
            case 9:     // BVS (VS = V)
                sprintf ( buf, " bvs%s", size);
                break;
            case 10: // BPL (PL = !N)
                sprintf ( buf, " bpl%s", size);
                break;
            case 11: // BMI (MI = N)
                sprintf ( buf, " bmi%s", size);
                break;
            case 12: // BGE (GE = N & V | !N & !V)
                sprintf ( buf, " bge%s", size);
                break;
            case 13: // BLT (LT = N & !V | !N & V)
                sprintf ( buf, " blt%s", size);
                break;
            case 14: // BGT (GT = N & V & !Z | !N & !V & !Z)
                sprintf ( buf, " bgt%s", size);
                break;
            case 15: // BLE (LE = Z | N & !V | !N & V)
                sprintf ( buf, " ble%s", size);
                break;
        }
    }

    return _address;
}

unsigned long M68008::tS_bits( unsigned long _address, char *buf, char* Instr, const char* Arg1 )
```

```
{
   //
   // Alan Donnelly (16/04/02)
   //     auxilary method for all  bit  testing/setting
   //     and changing instructions.
   //

   unsigned char EaMode = GET_TINSTRBITS(0x38, 3);
   unsigned char EaReg = GET_TINSTRBITS(0x7, 3);

   // (d16, An)
   signed short int Disp16;

   // Disp8(An, Xn)
   signed char Disp8;
   unsigned char Xn;
   bool Wl;

   switch( EaMode )
   {
      case 0:      // Dn.l
         sprintf ( buf, "%s.l %s,d%X", Instr, Arg1, EaReg );
         break;
      case 1:      // An (n/a)
         illegal ();
         break;
      case 2:      // (An).b
         sprintf ( buf, "%s.b %s,(a%X)", Instr, Arg1, EaReg );
         break;
      case 3:      // (An)+.b
         sprintf ( buf, "%s.b %s,(a%X)+", Instr, Arg1, EaReg );
      case 4:      // -(An).b
         sprintf ( buf, "%s.b %s,-(a%X)", Instr, Arg1, EaReg );
         break;
      case 5:      // (d16, An).b
         Disp16 = memory->GetWord( _address + 2 );
         if ( decImm )sprintf ( buf, "%s.b %s,%d(a%X)", Instr, Arg1, Disp16, EaReg );
         else  sprintf ( buf, "%s.b %s,$%X(a%X)", Instr, Arg1, Disp16, EaReg );

         // Update the PC
         _address += 2;

         break;
      case 6:      // d8(An, Xn).b
         Disp8 = memory->GetByte( _address + 3 );
         Xn = GETBITS( _address + 2, 0xC0, 4 );
         Wl = memory->GetByte( _address + 2 )& 0x1;

         if ( Wl )
            if ( decImm )sprintf ( buf, "%s.b %s,%d(a%X,d%X.w)", Instr, Arg1, Disp8, Xn, EaReg );
            else  sprintf ( buf, "%s.b %s,$%X(a%X,d%X.w)", Instr, Arg1, Disp8, Xn, EaReg );
         else
```

```
            if ( decImm )sprintf ( buf, "%s.b %s,%d(a%X,d%X.l)", Instr, Arg1, Disp8, Xn, EaReg );
            else  sprintf ( buf, "%s.b %s,$%X(a%X,d%X.l)", Instr, Arg1, Disp8, Xn, EaReg );

         // Update the PC
         _address += 2;
         break;
      case 7:
         switch( EaMode )
         {
            case 0:  //(xxx).W.b
               sprintf ( buf, "%s.b %s,$%lX", Instr, Arg1, memory->GetLongword( _address + 2 ));

               // Update the PC
               _address += 4;

               break;
            case 1:  //(xxx).L.b
               sprintf ( buf, "%s.b %s,$%lX", Instr, Arg1, memory->GetLongword( _address + 2 ));

               // Update the PC
               _address += 4;

               break;
            default:
               illegal ();
               break;
         }

         break;
   }

   // Update _address
   _address += 2;

   return _address;
}

unsigned long M68008::tS_bchgStat( unsigned long _address, char *buf )
{
   //
   // Alan Donnelly (02/03/02)
   // Uses btst, bset and bclr to change the bit
   //

   char Arg1 [ 32 ];
   if ( decImm )sprintf ( Arg1, "#%d", memory->GetByte( _address + 3 )% 8 );
   else  sprintf ( Arg1, "#$%X", memory->GetByte( _address + 3 )% 8 );
// unsigned long SavedPC = 0; // save PC (see bchgDyn())

   // Update _address first to account for immediate byte that follows instruction.
   _address += 2;
```

```
    _address = tS_bits( _address, buf, "bchg", Arg1 );

    return _address;

}

unsigned long M68008::tS_bchgDyn( unsigned long _address, char *buf )
{
    //
    // Alan Donnelly (02/03/02)
    // Uses btst, bset and bclr to change the bit
    //

    unsigned char Reg = GET_TINSTRBITS(0xE00, 9);
    char Arg1[ 32 ];

    sprintf( Arg1, "d%X", Reg );

    _address = tS_bits( _address, buf, "bchg", Arg1 );

    // PC updated by auxilary method

    return _address;
}

unsigned long M68008::tS_bclrStat( unsigned long _address, char *buf )
{
    //
    // Alan Donnelly (02/03/02)
    // uses bclr to clear the bit
    //

    char Arg1 [ 32 ];
    if( decImm )sprintf( Arg1, "#%d", memory->GetByte( _address + 3 )% 8 );
    else sprintf( Arg1, "#$%X", memory->GetByte( _address + 3 )% 8 );

    // Update _address first to account for immediate byte that follows instruction.
    _address += 2;

    return tS_bits( _address, buf, "bclr", Arg1 );

}

unsigned long M68008::tS_bclrDyn( unsigned long _address, char *buf )
{
    //
    // Alan Donnelly (02/03/02)
    // uses bclr to clear the bit

    unsigned char Reg = GET_TINSTRBITS(0xE00, 9);
    char Arg1 [ 32 ];
```

```
    sprintf ( Arg1, "d%X", Reg );

    // PC updated by auxilary methods

    return tS_bits( _address, buf, "bclr", Arg1 );
}


unsigned long M68008::tS_bkpt( unsigned long _address, char *buf )
{
     //
    // Gerard Whyte
     //   For the MC68000 and MC68008, the breakpoint cycle is not run,
    // but an illegal instruction exception is taken.
     //
    unsigned char bkptnum = GETBITS( _address, 7, 0 );

    sprintf ( buf, "bkpt $%X", bkptnum );
    _address += 2;

    return _address;
}

unsigned long M68008::tS_bra( unsigned long _address, char *buf )
{
    //
    // Alan Donnelly (02/03/02)
    // (Note: CCR not affected)
    //

    signed char Disp8 = GET_TINSTRBITS(0xFF, 0);
    signed short int Disp16 = 0;

    if( Disp8 == -1 )  // 32-bit disp not supported
    {
        sprintf ( buf, "32-Bit Displacement not supported" );
        _address += 6;
    }
    else if ( Disp8 == 0 ) // 16-bit imm. disp if 8-bit disp == 0
    {
        Disp16 = memory->GetWord( _address + 2 );
        sprintf ( buf, "bra.w $%lX", Disp16 + _address + 2);
        _address += 4;
    }
    else                // 8-bit displacement
    {
        sprintf ( buf,"bra.b $%lX", Disp8 + _address + 2);
        _address += 2;
    }
    return _address;
}
```

```
unsigned long M68008::tS_bsetStat( unsigned long _address, char *buf )
{
   //
   // Alan Donnelly (02/03/02)
   // uses bset to set the bit
   //

   char Arg1 [ 32 ];

   if( decImm )sprintf( Arg1, "#%d", memory->GetByte( _address + 3 )% 8 );
   else sprintf( Arg1, "#$%X", memory->GetByte( _address + 3 )% 8 );

   // Update _address first to account for immediate byte that follows instruction.
   _address += 2;

   return tS_bits( _address, buf, "bset", Arg1 );
}

unsigned long M68008::tS_bsetDyn( unsigned long _address, char *buf )
{
   //
   // Alan Donnelly (02/03/02)
   // uses bset to set the bit
   //

   unsigned char Reg = GET_TINSTRBITS(0xE00, 9);
   char Arg1 [ 32 ];

   sprintf( Arg1, "d%X", Reg );

   // PC updated by auxilary method

   return tS_bits( _address, buf, "bset", Arg1 );
}

unsigned long M68008::tS_bsr( unsigned long _address, char *buf )
{
   //
   // Alan Donnelly (02/03/02)
   // (Note: CCR not affected)
   //

   signed char Disp8 = GET_TINSTRBITS(0xFF, 0);
   signed short int Disp16 = 0;

   if( Disp8 == -1 )  // 32-bit disp not supported
   {
      sprintf( buf, "32-Bit_Displacement_not_supported_for_bsr" );
      _address += 6;
   }
   else if ( Disp8 == 0 ) // 16-bit imm. disp if 8-bit disp == 0
   {
```

```
        Disp16 = memory->GetWord( _address + 2 );
        sprintf ( buf, "bsr.w $%lX", Disp16 + _address + 2 );
        _address += 4;
    }
    else                    // 8-bit displacement
    {
        sprintf ( buf,"bsr.b $%lX", Disp8 + _address + 2 );
        _address += 2;
    }

    // _address already updated

    return _address;
}

unsigned long M68008::tS_btstStat( unsigned long _address, char *buf )
{
    //
    // Alan Donnelly (02/03/02)
    // uses btst to test the bit

    char Arg1 [ 32 ];
    if ( decImm )sprintf ( Arg1,"#%d", memory->GetByte( _address + 3 )% 8);
    else  sprintf ( Arg1,"#$%X", memory->GetByte( _address + 3 )% 8);

    // Update _address first to account for immediate byte that follows instruction.
    _address += 2;

    return tS_bits( _address, buf, "bset", Arg1 );
}

unsigned long M68008::tS_btstDyn( unsigned long _address, char *buf )
{
    //
    // Alan Donnelly (02/03/02)
    // uses btst to test the bit

    char Arg1 [ 32 ];
    unsigned char Reg = GET_TINSTRBITS(0xE00, 9);

    sprintf ( Arg1, "d%X", Reg );

    // PC updated by auxilary method

    return tS_bits( _address, buf, "btst", Arg1 );
}

unsigned long M68008::tS_chk( unsigned long _address, char *buf )
{
    //
    // Gerard Whyte
    // Only need to do WORD operation
```

```
//
unsigned char Reg = GET_TINSTRBITS(0xE00, 9);
unsigned char EaMode = GET_TINSTRBITS(0x38, 3);
unsigned char EaReg = GET_TINSTRBITS(0x7, 0);
signed short int upper;

char EA[ 32 ];

// (d16, An)
signed short int Disp16;

// Disp8(An, Xn)
signed char Disp8 = 0;
unsigned char Xn = 0;
unsigned char Wl;

switch ( EaMode )
{
   case 0: // Dn
      sprintf ( EA, "d%X", EaReg );
         break;
     case 2: // (An)
      sprintf ( EA, "(a%X)", EaReg );
     break;
     case 3: // (An)+
      sprintf ( EA, "(a%X)+", EaReg );
         break;
     case 4: // -(An)
         sprintf ( EA, "-(a%X)", EaReg );
         break;
     case 5: // Disp16(An)
         Disp16 = memory->GetWord( _address + 2 );
         if ( decImm )sprintf ( EA, "%d(a%X)", Disp16, EaReg );
         else  sprintf ( EA, "$%X(a%X)", Disp16, EaReg );
      _address += 2;
         break;
     case 6: // Disp8(An, Xn)
      Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
        Xn = ( memory->GetByte( _address + 2 )& 0xC0 )>> 4;
       Wl = memory->GetByte( _address + 2 )& 0x1;

         if ( Wl )     // word index
         if ( decImm )sprintf ( EA, "%d(a%X,d%X.w)", Disp8, EaReg, Xn );
         else  sprintf ( EA, "$%X(a%X,d%X.w)", Disp8, EaReg, Xn );
         else
         if ( decImm )sprintf ( EA, "%d(a%X,d%X.l)", Disp8, EaReg, Xn );
         else  sprintf ( EA, "$%X(a%X,d%X.l)", Disp8, EaReg, Xn );
      _address += 2;
         break;
   case 7:
      switch( EaReg )
        {
```

```
            case 0:          // (www).W
                sprintf ( EA, "$%X", memory->GetWord( _address + 2 ));
                _address += 2;
                    break;
                case 1:          // (www).L
                sprintf ( EA, "$%lX", memory->GetLongword( _address + 2 ));
                _address += 4;
                    break;
                case 2:          // (d16, PC).b
                _address += 2;
                Disp16 = memory->GetWord( _address );
                sprintf ( EA, "($%X,PC).b", Disp16 );
                    break;
                case 3:          // d8(PC, Xn)
                _address += 2;
                    Disp8 = memory->GetByte( _address + 1 ); //skip 1st byte of imm. operand
                    Xn = GETBITS( _address, 0xC0, 4 );
                    Wl = memory->GetByte( _address )& 0x1;

                    if ( Wl )       // word index
                  if ( decImm )sprintf ( EA, "%d(PC,d%X.w)", Disp8, Xn );
                  else  sprintf ( EA, "$%X(PC,d%X.w)", Disp8, Xn );
                    else            // long index
                  if ( decImm )sprintf ( EA, "%d(PC,d%X.l)", Disp8, Xn );
                  else  sprintf ( EA, "$%X(PC,d%X.l)", Disp8, Xn );
                _address += 2;
                    break;
                case 4:          // #<data>
                _address += 2;
                upper = memory->GetWord( _address );
                if ( decImm )sprintf ( EA, "#%d", upper );
                else  sprintf ( EA, "#$%X", upper );
                break;
                 case 5:
                 case 6:
                 case 7:
                break;
            }
        }
    if  ( ( EaMode == 1 )|| ( EaMode == 7 && EaReg > 4 ))
        sprintf ( buf, " Illegal instruction . Get a better assembler!" );
    else
        sprintf ( buf, "chk.w %s,d%X", EA, Reg );
    _address += 2;
    return _address;
}

unsigned long M68008::tS_clr( unsigned long _address, char *buf )
{
    //
    // Alan Donnelly and Gerard Whyte (25/02/2002)
    //
```

```
//
// change d[Dn]++ for word and long operations
//

unsigned char Size = GET_TINSTRBITS(0xC0, 6);
unsigned char EaMode = GET_TINSTRBITS(0x38, 3);
unsigned char EaReg = GET_TINSTRBITS(0x7, 0);

// (d16, An)
signed short int Disp16;

// Disp8(An, Xn)
signed char Disp8;
unsigned char Xn;
unsigned char Wl;

unsigned long int address;

char pmtr[ 32 ];

// Clear the  register
switch ( ( Size << 3 ) + EaMode )
{
   // BYTE
   case 0:          // Dn.b
      sprintf ( pmtr, ".b␣d%X", EaReg );
      break;
   case 1:          // An.b (doesn't apply)
      break;
   case 2:          // (An).b
      sprintf ( pmtr, ".b␣(a%X)", EaReg );
      break;
   case 3:          // (An)+.b
      sprintf ( pmtr, ".b␣(a%X)+", EaReg );
      break;
   case 4:          // -(An).b
      sprintf ( pmtr, ".b␣-(a%X)", EaReg );
      break;
   case 5:          // Disp16(An).b
      _address += 2;
      Disp16 = memory->GetWord( _address );
      if ( decImm )sprintf ( pmtr, ".b␣%d(a%X)", Disp16, EaReg );
      else  sprintf ( pmtr, ".b␣$%X(a%X)", Disp16, EaReg );
      break;
   case 6:          // Disp8(An, Xn).b
      Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
      Xn = ( memory->GetByte( _address + 2 )& 0xC0 )>> 4;
      Wl = memory->GetByte( _address + 2 )& 0x1;

      if ( Wl )     // word index
         if ( decImm )sprintf ( pmtr, ".b␣%d(a%X,d%X.w)", Disp8, EaReg, Xn );
```

```
        else sprintf( pmtr, ".b $%X(a%X,d%X.w)", Disp8, EaReg, Xn );
    else
        if( decImm )sprintf( pmtr, ".b %d(a%X,d%X.l)", Disp8, EaReg, Xn );
        else sprintf( pmtr, ".b $%X(a%X,d%X.l)", Disp8, EaReg, Xn );
    _address += 2;
    break;
case 7:
    if( !EaReg )        // (xxx).W.b
    {
        address = memory->GetWord( _address + 2 );
        _address += 2;
    }
    else            // (xxx).L.b
    {
        address = memory->GetLongword( _address + 2 );
        _address += 4;
    }
    sprintf( pmtr, ".b $%lX", address );
    break;
// WORD
case 8:
    sprintf( pmtr, ".w d%X", EaReg );
    break;
case 9:             // An.w
    break;
case 10:            // (An).w
    sprintf( pmtr, ".w (a%X)", EaReg );
    break;
case 11:            // (An)+.w
    sprintf( pmtr, ".w (a%X)+", EaReg );
    break;
case 12:            // -(An).w
    sprintf( pmtr, ".w -(a%X)", EaReg );
    break;
case 13:            // Disp16(An).w
    _address += 2;
    Disp16 = memory->GetWord( _address );
    if( decImm )sprintf( pmtr, ".w %d(a%X)", Disp16, EaReg );
    else sprintf( pmtr, ".w $%X(a%X)", Disp16, EaReg );
    break;
case 14:            // Disp8(An, Xn).w
    Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
    Xn = ( memory->GetByte( _address + 2 )& 0xC0 )>> 4;
    Wl = memory->GetByte( _address + 2 )& 0x1;

    if( Wl )    // word index
        if( decImm )sprintf( pmtr, ".w %d(a%X,d%X.w)", Disp8, EaReg, Xn );
        else sprintf( pmtr, ".w $%X(a%X,d%X.w)", Disp8, EaReg, Xn );
    else
        if( decImm )sprintf( pmtr, ".w %d(a%X,d%X.l)", Disp8, EaReg, Xn );
        else sprintf( pmtr, ".w $%X(a%X,d%X.l)", Disp8, EaReg, Xn );
    _address += 2;
```

```
        break;
    case 15:
        if ( !EaReg )       // (xxx).W.w
        {
            address = memory->GetWord( _address + 2 );
            _address += 2;
        }
        else            // (xxx).L.w
        {
            address = memory->GetLongword( _address + 2 );
            _address += 4;
        }
        sprintf ( pmtr, ".w $%lX", address );
        break;
    // LONGWORD
    case 16:        // Dn.l
        sprintf ( pmtr, ".l d%X", EaReg );
        break;
    case 17:        // An.l
        break;
    case 18:        // (An).l
        sprintf ( pmtr, ".l (a%X)", EaReg );
        break;
    case 19:        // (An)+.l
        sprintf ( pmtr, ".l (a%X)+", EaReg );
        break;
    case 20:        // -(An).b
        sprintf ( pmtr, ".l -(a%X)", EaReg );
        break;
    case 21:        // Disp16(An).l
        _address += 2;
        Disp16 = memory->GetWord( _address );
        if ( decImm )sprintf ( pmtr, ".l %d(a%X)", Disp16, EaReg );
        else  sprintf ( pmtr, ".l $%X(a%X)", Disp16, EaReg );
        break;
    case 22:        // Disp8(An, Xn).l
        Disp8 = memory->GetByte( _address + 3 ); //skip 1st byte of imm. operand
        Xn = ( memory->GetByte( _address + 2 )& 0xC0 )>> 4;
        Wl = memory->GetByte( _address + 2 )& 0x1;

        if ( Wl )     // word index
            if ( decImm )sprintf ( pmtr, ".l %d(a%X,d%X.w)", Disp8, EaReg, Xn );
            else  sprintf ( pmtr, ".l $%X(a%X,d%X.w)", Disp8, EaReg, Xn );
        else
            if ( decImm )sprintf ( pmtr, ".l %d(a%X,d%X.l)", Disp8, EaReg, Xn );
            else  sprintf ( pmtr, ".l $%X(a%X,d%X.l)", Disp8, EaReg, Xn );
        _address += 2;
        break;
    case 23:
        if  ( !EaReg )      // (xxx).W.l
        {
            address = memory->GetWord( _address + 2 );
```

```
            _address += 2;
        }
        else          // (xxx).L.l
        {
            address = memory->GetLongword( _address + 2 );
            _address += 4;
        }
        sprintf ( pmtr, ".l $%lX", address );
        break;
    }

    if ( EaMode == 1 )
        sprintf ( buf, " Illegal instruction . Get a better assembler!" );
    else
        sprintf ( buf, " clr%s", pmtr );

    _address += 2;
    return _address;
}

unsigned long M68008::tS_cmp( unsigned long _address, char *buf )
{
    int mode, opmode, reg, eaReg, xn;
    unsigned long addr;

    reg    = GETBITS( _address, 0x0E00, 9 );
    opmode = GETBITS( _address, 0x01C0, 6 );
    mode   = GETBITS( _address, 0x0038, 3 );
    eaReg  = GETBITS( _address, 0x0007, 0 );

    // This instruction gets executed for eor() and cmpa() as well
    // Rectifying this here
    if ( ( opmode & 3 )== 3 )
        return tS_cmpa( _address, buf );
    else if ( opmode >= 4 && mode == 1 )
        return tS_cmpm( _address, buf );
    else if ( opmode >= 4 )
        return tS_eor( _address, buf );
    else
    {
        // dn,dn
        if ( mode == 0 )
        {
            if ( opmode == 0 )
            {
                sprintf ( buf, " cmp.b d%X,d%X", eaReg, reg );
            }
            if ( opmode == 1 )
            {
                sprintf ( buf, " cmp.w d%X,d%X", eaReg, reg );
            }
            if ( opmode == 2 )
```

```
    {
        sprintf ( buf, "cmp.l␣d%X,d%X", eaReg, reg );
    }
    _address += 2;
}

// an,dn
if ( mode == 1 )
{
    if ( opmode == 1 )
    {
        sprintf ( buf, "cmp.w␣a%X,d%X", eaReg, reg );
    }
    if ( opmode == 2 )
    {
        sprintf ( buf, "cmp.l␣a%X,d%X", eaReg, reg );
    }
    _address += 2;
}

// (an),dn
if ( mode == 2 )
{
    if ( opmode == 0 )
    {
        sprintf ( buf, "cmp.b␣(a%X),d%X", eaReg, reg );
    }
    if ( opmode == 1 )
    {
        sprintf ( buf, "cmp.w␣(a%X),d%X", eaReg, reg );
    }
    if ( opmode == 2 )
    {
        sprintf ( buf, "cmp.l␣(a%X),d%X", eaReg, reg );
    }
    _address += 2;
}

// (an)+,dn
if ( mode == 3 )
{
    if ( opmode == 0 )
    {
        sprintf ( buf, "cmp.b␣(a%X)+,d%X", eaReg, reg );
    }
    if ( opmode == 1 )
    {
        sprintf ( buf, "cmp.w␣(a%X)+,d%X", eaReg, reg );
    }
    if ( opmode == 2 )
    {
        sprintf ( buf, "cmp.l␣(a%X)+,d%X", eaReg, reg );
```

```
    }

    _address += 2;
}

// -(an),dn
if( mode == 4 )
{
    if( opmode == 0 )
    {
        sprintf( buf, "cmp.b -(a%X),d%X", eaReg, reg );
    }
    if( opmode == 1 )
    {
        sprintf( buf, "cmp.w -(a%X),d%X", eaReg, reg );
    }
    if( opmode == 2 )
    {
        sprintf( buf, "cmp.l -(a%X),d%X", eaReg, reg );
    }

    _address += 2;

}

// d16(an),dn
if( mode == 5 )
{
    unsigned long addr = (signed short int)memory->GetWord( _address + 2 );
    if( opmode == 0 )
    {
        if( decImm )sprintf( buf, "cmp.b %ld(a%X),d%X", addr, eaReg, reg );
        else sprintf( buf, "cmp.b $%lX(a%X),d%X", addr, eaReg, reg );
    }
    if( opmode == 1 )
    {
        if( decImm )sprintf( buf, "cmp.w %ld(a%X),d%X", addr, eaReg, reg );
        else sprintf( buf, "cmp.w $%lX(a%X),d%X", addr, eaReg, reg );
    }
    if( opmode == 2 )
    {
        if( decImm )sprintf( buf, "cmp.l %ld(a%X),d%X", addr, eaReg, reg );
        else sprintf( buf, "cmp.l $%lX(a%X),d%X", addr, eaReg, reg );
    }
    _address += 4;
}

// d8(an,Xn),dn
if( mode == 6 )
{
    addr = (signed char)memory->GetByte( _address + 3 );
    xn = GETBITS( _address + 2, 0xF000, 12 );
```

```
    if ( opmode == 0 )
    {
        if ( decImm )sprintf ( buf, "cmp.b #%ld(a%X,d%X),d%X", addr, eaReg, xn, reg );
        else  sprintf ( buf, "cmp.b #$%lX(a%X,d%X),d%X", addr, eaReg, xn, reg );
    }
    if ( opmode == 1 )
    {
        if ( decImm )sprintf ( buf, "cmp.w #%ld(a%X,d%X),d%X", addr, eaReg, xn, reg );
        else  sprintf ( buf, "cmp.w #$%lX(a%X,d%X),d%X", addr, eaReg, xn, reg );
    }
    if ( opmode == 2 )
    {
        if ( decImm )sprintf ( buf, "cmp.l #%ld(a%X,d%X),d%X", addr, eaReg, xn, reg );
        else  sprintf ( buf, "cmp.l #$%lX(a%X,d%X),d%X", addr, eaReg, xn, reg );
    }
    _address += 4;
}

//(XXX).W,dn
if ( mode == 7 && eaReg < 2 )
{
    if ( eaReg == 0 )
    {
        addr = memory->GetWord( _address + 2 );
        _address += 4;
    }
    else
    {
        addr = memory->GetLongword( _address + 2 );
        _address += 6;
    }
    if ( opmode == 0 )
    {
        sprintf ( buf, "cmp.b #$%lX,d%X", addr, reg );
    }
    if ( opmode == 1 )
    {
        sprintf ( buf, "cmp.w #$%lX,d%X", addr, reg );
    }
    if ( opmode == 2 )
    {
        sprintf ( buf, "cmp.l #$%lX,d%X", addr, reg );
    }
}

// #<data>,dn
if ( mode == 7 && eaReg == 4)
{
    if ( opmode == 0 )// Byte
    {
        if ( decImm )sprintf ( buf, "cmp.b #%d,d%X", memory->GetByte( _address + 3 ), reg );
```

```
            else sprintf( buf, "cmp.b #$%X,d%X", memory->GetByte( _address + 3 ), reg );
            _address += 4;
        }
        if( opmode == 1 )// Word
        {
            if( decImm )sprintf( buf, "cmp.w #%d,d%X", memory->GetWord( _address + 2 ), reg );
            else sprintf( buf, "cmp.w #$%X,d%X", memory->GetWord( _address + 2 ), reg );
            _address += 4;
        }
        if( opmode == 2 )// Longword
        {
            if( decImm )sprintf( buf, "cmp.l #%ld,d%X", memory->GetLongword( _address + 2 ), reg );
            else sprintf( buf, "cmp.l #$%lX,d%X", memory->GetLongword( _address + 2 ), reg );
            _address += 6;
        }
    }

    // OMITTED
    // ( D16, PC )
    // ( d8, PC, Xn )

    return _address;
  }
}

unsigned long M68008::tS_cmpa( unsigned long _address, char *buf )
{
  int mode, opmode, reg, eaReg, xn;
  unsigned long addr;
  ADDRESS_REGISTER( imm );

  reg = GETBITS( _address, 0x0E00, 9 );
  opmode = GETBITS( _address, 0x01C0, 6 );
  mode = GETBITS( _address, 0x0038, 3 );
  eaReg = GETBITS( _address, 0x0007, 0 );

  // dn,an
  if( mode == 0 )
  {
      if( opmode == 3 )
      {
          sprintf( buf, "cmpa.w d%X,a%X", eaReg, reg );
      }
      if( opmode == 7 )
      {
          sprintf( buf, "cmpa.l d%X,a%X", eaReg, reg );
      }
      _address += 2;
  }
  // an, an
  if( mode == 1 )
  {
```

```
    if ( opmode == 3 )
    {
        sprintf ( buf, "cmpa.w a%X,a%X", eaReg, reg );
    }
    if ( opmode == 7 )
    {
        sprintf ( buf, "cmpa.l a%X,a%X", eaReg, reg );
    }
    address += 2;
}
// ( an ), an
if ( mode == 2 )
{
    if ( opmode == 3 )
    {
        sprintf ( buf, "cmpa.w (a%X),a%X", eaReg, reg );
    }
    if ( opmode == 7 )
    {
        sprintf ( buf, "cmpa.l (a%X),a%X", eaReg, reg );
    }
    address += 2;
}
// ( an )+, an
if ( mode == 3 )
{
    if ( opmode == 3 )
    {
        sprintf ( buf, "cmpa.w (a%X)+,a%X", eaReg, reg );
    }
    if ( opmode == 7 )
    {
        sprintf ( buf, "cmpa.l (a%X)+,a%X", eaReg, reg );
    }
    address += 2;
}
// -( an ), an
if ( mode == 4 )
{
    if ( opmode == 3 )
    {
        sprintf ( buf, "cmpa.w -(a%X),a%X", eaReg, reg );
    }
    if ( opmode == 7 )
    {
        sprintf ( buf, "cmpa.l -(a%X),a%X", eaReg, reg );
    }
    address += 2;
}
// d16(an), an
if ( mode == 5 )
{
```

```
    addr = (signed short int)memory->GetWord( _address + 2 );
    if ( opmode == 3 )
    {
        if ( decImm )sprintf ( buf, "cmpa.w %ld(a%X),a%X", addr, eaReg, reg );
        else  sprintf ( buf, "cmpa.w $%lX(a%X),a%X", addr, eaReg, reg );
    }
    if ( opmode == 7 )
    {
        if ( decImm )sprintf ( buf, "cmpa.l %ld(a%X),a%X", addr, eaReg, reg );
        else  sprintf ( buf, "cmpa.l $%lX(a%X),a%X", addr, eaReg, reg );
    }
    _address += 4;
}
// d8(an,xn), an
if ( mode == 6 )
{
    addr = (signed char)memory->GetByte( _address + 3 );
    xn = GETBITS( _address + 2, 0xf000, 12 );
    if ( opmode == 3 )
    {
        if ( decImm )sprintf ( buf, "cmpa.w %ld(a%X,d%X),a%X", addr, eaReg, xn, reg );
        else  sprintf ( buf, "cmpa.w $%lX(a%X,d%X),a%X", addr, eaReg, xn, reg );
    }
    if ( opmode == 7 )
    {
        if ( decImm )sprintf ( buf, "cmpa.l %ld(a%X,d%X),a%X", addr, eaReg, xn, reg );
        else  sprintf ( buf, "cmpa.l $%lX(a%X,d%X),a%X", addr, eaReg, xn, reg );
    }
    _address += 4;
}

// #<data>,an
if ( ( mode == 7 )&& ( eaReg == 4 ))
{
    if ( opmode == 3 )
    {
        imm.w = (signed short int)memory->GetWord( _address + 2 );
        if ( decImm )sprintf ( buf, "cmpa.w #%d,a%X", imm.w, reg );
        else  sprintf ( buf, "cmpa.w #$%X,a%X", imm.w, reg );
        _address += 4;
    }
    if ( opmode == 7 )
    {
        imm.l = (signed long int)memory->GetLongword( _address + 2 );
        if ( decImm )sprintf ( buf, "cmpa.l #%ld,a%X", imm.l, reg );
        else  sprintf ( buf, "cmpa.l #$%lX,a%X", imm.l, reg );
        _address += 6;
    }
    return _address;
}

//(XXX).W, an
```

```
    if ( mode == 7 )
    {
        if ( eaReg == 0 )
        {
            addr = memory->GetWord( _address + 2 );
            _address += 4;
        }
        else
        {
            //(XXX).L, an
            addr = memory->GetLongword( _address + 2 );
            _address += 6;
        }
        if ( opmode == 3 )
        {
            sprintf ( buf, "cmpa.w $%lX,a%X", addr, reg );
        }
        if ( opmode == 7 )
        {
            sprintf ( buf, "cmpa.l $%lX,a%X", addr, reg );
        }
    }
    return _address;
}

unsigned long M68008::tS_cmpi( unsigned long _address, char *buf )
{
    int  size , mode, reg, xn;
    unsigned long int displ;
    DATA_REGISTER( imm );

    size = GETBITS( _address, 0x00C0, 6 );
    mode = GETBITS( _address, 0x0038, 3 );
    reg = GETBITS( _address, 0x0007, 0 );

    // #data,dn
    if ( mode == 0 )
    {
        if ( size == 0 )
        {
            imm.b = memory->GetByte( _address + 3 );
            if ( decImm )sprintf ( buf, "cmpi.b #%d,d%X", imm.b, reg );
            else  sprintf ( buf, "cmpi.b #$%X,d%X", imm.b, reg );
            _address += 4;
        }
        if ( size == 1 )
        {
            imm.w = memory->GetWord( _address + 2 );
            if ( decImm )sprintf ( buf, "cmpi.w #%d,d%X", imm.w, reg );
            else  sprintf ( buf, "cmpi.w #$%X,d%X", imm.w, reg );
            _address += 4;
        }
```

```
    if ( size == 2 )
    {
        imm.l = memory->GetLongword( _address + 2 );
        if ( decImm )sprintf ( buf, "cmpi.l #%ld,d%X", imm.l, reg );
        else sprintf ( buf, "cmpi.l #$%lX,d%X", imm.l, reg );
        _address += 6;
    }
}

// #data,(an)
if ( mode == 2 )
{
    if ( size == 0 )
    {
        imm.b = memory->GetByte( _address + 3 );
        if ( decImm )sprintf ( buf, "cmpi.b #%d,(a%X)", imm.b, reg );
        else sprintf ( buf, "cmpi.b #$%X,(a%X)", imm.b, reg );
        _address += 4;
    }
    if ( size == 1 )
    {
        imm.w = memory->GetWord( _address + 2 );
        if ( decImm )sprintf ( buf, "cmpi.w #%d,(a%X)", imm.w, reg );
        else sprintf ( buf, "cmpi.w #$%X,(a%X)", imm.w, reg );
        _address += 4;
    }
    if ( size == 2 )
    {
        imm.l = memory->GetLongword( _address + 2 );
        if ( decImm )sprintf ( buf, "cmpi.l #%ld,(a%X)", imm.l, reg );
        else sprintf ( buf, "cmpi.l #$%lX,(a%X)", imm.l, reg );
        _address += 6;
    }
}

// #data, (an)+
if ( mode == 3 )
{
    if ( size == 0 )
    {
        imm.b = memory->GetByte( _address + 3 );
        if ( decImm )sprintf ( buf, "cmpi.b #%d,(a%X)+", imm.b, reg );
        else sprintf ( buf, "cmpi.b #$%X,(a%X)+", imm.b, reg );
        _address += 4;
    }
    if ( size == 1 )
    {
        imm.w = memory->GetWord( _address + 2 );
        if ( decImm )sprintf ( buf, "cmpi.w #%d,(a%X)+", imm.w, reg );
        else sprintf ( buf, "cmpi.w #$%X,(a%X)+", imm.w, reg );
        _address += 4;
    }
```

```
    if( size == 2 )
    {
        imm.l = memory->GetLongword( _address + 2 );
        if( decImm )sprintf( buf, "cmpi.l #%ld,(a%X)+", imm.l, reg );
        else sprintf( buf, "cmpi.l #$%lX,(a%X)+", imm.l, reg );
        _address += 6;
    }
}

// #data, -(an)
if( mode == 4 )
{
    if( size == 0 )
    {
        imm.b = memory->GetByte( _address + 3 );
        if( decImm )sprintf( buf, "cmpi.b #%d,-(a%X)", imm.b, reg );
        else sprintf( buf, "cmpi.b #$%X,-(a%X)", imm.b, reg );
        _address += 4;
    }
    if( size == 1 )
    {
        imm.w = memory->GetWord( _address + 2 );
        if( decImm )sprintf( buf, "cmpi.w #%d,-(a%X)", imm.w, reg );
        else sprintf( buf, "cmpi.w #$%X,-(a%X)", imm.w, reg );
        _address += 4;
    }
    if( size == 2 )
    {
        imm.l = memory->GetLongword( _address + 2 );
        if( decImm )sprintf( buf, "cmpi.l #%ld,-(a%X)", imm.l, reg );
        else sprintf( buf, "cmpi.l #$%lX,-(a%X)", imm.l, reg );
        _address += 6;
    }
}

// #data, d16(an)
if( mode == 5 )
{
    if( size == 0 )
    {
        displ = (signed short int)memory->GetWord( _address + 4 );
        imm.b = memory->GetByte( _address + 3 );
        if( decImm )sprintf( buf, "cmpi.b #$%X,%ld(a%X)", imm.b, displ, reg );
        else sprintf( buf, "cmpi.b #$%X,$%lX(a%X)", imm.b, displ, reg );
        _address += 6;
    }
    if( size == 1 )
    {
        displ = (signed short int)memory->GetWord( _address + 4 );
        imm.w = memory->GetWord( _address + 2 );
        if( decImm )sprintf( buf, "cmpi.w #$%X,%ld(a%X)", imm.w, displ, reg );
        else sprintf( buf, "cmpi.w #$%X,$%lX(a%X)", imm.w, displ, reg );
```

```
        _address += 6;
    }
    if( size == 2 )
    {
        displ = (signed short int)memory->GetWord( _address + 6 );
        imm.l = memory->GetLongword( _address + 2 );
        if( decImm )sprintf( buf, "cmpi.l #$%lX,%ld(a%X)", imm.l, displ, reg );
        else sprintf( buf, "cmpi.l #$%lX,$%lX(a%X)", imm.l, displ, reg );
        _address += 8;
    }
}

// #data, d8(an,xn)
if( mode == 6 )
{
    xn = GETBITS( _address + 4, 0xF000, 12 );
    if( size == 0 )
    {
        displ = (signed char)memory->GetByte( _address + 5 );
        imm.b = memory->GetByte( _address + 3 );
        if( decImm )sprintf( buf, "cmpi.b #$%X,%ld(a%X,d%X)", imm.b, displ, reg, xn );
        else sprintf( buf, "cmpi.b #$%X,$%lX(a%X,d%X)", imm.b, displ, reg, xn );
        _address += 6;
    }
    if( size == 1 )
    {
        displ = (signed char)memory->GetByte( _address + 5 );
        imm.w = memory->GetWord( _address + 2 );
        if( decImm )sprintf( buf, "cmpi.w #$%X,%ld(a%X,d%X)", imm.w, displ, reg, xn );
        else sprintf( buf, "cmpi.w #$%X,$%lX(a%X,d%X)", imm.w, displ, reg, xn );
        _address += 6;
    }
    if( size == 2 )
    {
        displ = (signed char)memory->GetByte( _address + 7 );
        imm.l = memory->GetLongword( _address + 2 );
        if( decImm )sprintf( buf, "cmpi.l #$%lX,%ld(a%X,d%X)", imm.l, displ, reg, xn );
        else sprintf( buf, "cmpi.l #$%lX,$%lX(a%X,d%X)", imm.l, displ, reg, xn );
        _address += 8;
    }
}

if( mode == 7 )
{
    // #data, (xxx).w
    if( reg == 0 )
    {
        if( size == 0 )
        {
            displ = memory->GetWord( _address + 4 );
            imm.b = memory->GetByte( _address + 3 );
            if( decImm )sprintf( buf, "cmpi.b #%d,$%lX", imm.b, displ );
```

```
            else sprintf ( buf, "cmpi.b #$%X,$%lX", imm.b, displ );
            _address += 6;
        }
        if ( size == 1 )
        {
            displ = memory->GetWord( _address + 4 );
            imm.w = memory->GetWord( _address + 2 );
            if ( decImm )sprintf ( buf, "cmpi.w #%d,$%lX", imm.w, displ );
            else sprintf ( buf, "cmpi.w #$%X,$%lX", imm.w, displ );
            _address += 6;
        }
        if ( size == 2 )
        {
            displ = memory->GetWord( _address + 6 );
            imm.l = memory->GetLongword( _address + 2 );
            if ( decImm )sprintf ( buf, "cmpi.l #%ld,$%lX", imm.l, displ );
            else sprintf ( buf, "cmpi.l #$%lX,$%lX", imm.l, displ );
            _address += 8;
        }
    }

    // #data, (xxx).l
    if ( reg == 1 )
    {
        if ( size == 0 )
        {
            displ = memory->GetLongword( _address + 4 );
            imm.b = memory->GetByte( _address + 3 );
            if ( decImm )sprintf ( buf, "cmpi.b #%d,$%lX", imm.b, displ );
            else sprintf ( buf, "cmpi.b #$%X,$%lX", imm.b, displ );
            _address += 8;
        }
        if ( size == 1 )
        {
            displ = memory->GetLongword( _address + 4 );
            imm.w = memory->GetWord( _address + 2 );
            if ( decImm )sprintf ( buf, "cmpi.w #%d,$%lX", imm.w, displ );
            else sprintf ( buf, "cmpi.w #$%X,$%lX", imm.w, displ );
            _address += 8;
        }
        if ( size == 2 )
        {
            displ = memory->GetLongword( _address + 6 );
            imm.l = memory->GetLongword( _address + 2 );
            if ( decImm )sprintf ( buf, "cmpi.l #%ld,$%lX", imm.l, displ );
            else sprintf ( buf, "cmpi.l #$%lX,$%lX", imm.l, displ );
            _address += 10;
        }
    }
  }
  return _address;
}
```

```
unsigned long M68008::tS_cmpm( unsigned long _address, char *buf )
{
    unsigned char Ax = GET_TINSTRBITS( 0xE00, 9 );
    unsigned char Ay = GET_TINSTRBITS( 0x7, 0 );

    unsigned char Size = GET_TINSTRBITS( 0xC0, 6 );
   char opsz[10];

    switch ( Size )
    {
      case 0:
          sprintf  ( opsz, ".b");
             break;
        case 1:
             sprintf  ( opsz, ".w");
             break;
        case 2:
             sprintf  ( opsz, ".l");
         break;
   }

   sprintf  ( buf, "cmpm%s (a%X)+,(a%X)+", opsz, Ay, Ax );

   _address += 2;
   return _address;
}
```

1.6 MInstrD-N.cpp

The execution methods of the instructions starting with $D \cdots N$.

```
/*
   Motorola 68008 Simulator
   (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
   Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

   This class represents the Motorola 68008 Microprocessor
   Instruction definitions D-N
*/

#include <stdio.h>
#include "M68008.h"
#define GETBITS( addr, mask, shift )(( memory->GetWord( addr )& mask )>> shift)
#define GETLONGBITS( addr, mask, shift )(( memory->GetLongword( addr )& mask )>> shift)
#define SETNZ( src ){ ccr.n = ( (signed char)src < 0 ); ccr.z = ( src == 0 ); }
#define SETNZL( src ){ ccr.n = ( (signed long)src < 0 ); ccr.z = ( src == 0 ); }
#define SETNZW( src ){ ccr.n = ( (signed short)src < 0 ); ccr.z = ( src == 0 ); }
#include <qmessagebox.h>

void M68008::dbxx()
{
```

```
short int reg = GETBITS( pc, 0x0007, 0);
short int condition = GETBITS( pc, 0x0f00, 8);
signed short int displ = memory->GetWord( pc + 2 );

// Condition codes listed on page 90
//
// T     True            0000
// F     False           0001
// HI    High            0010
// LS    Low or same     0011
// CC(HI) Carry Clear    0100
// CS(LO) Carry Set      0101
// NE    Not Equal       0110
// EQ    Equal           0111
// VC    Overflow clear  1000
// VS    Overflow set    1001
// PL    Plus            1010
// MI    Minus           1011
// GE    Greater or equal 1100
// LT    Less than       1101
// GT    Greater than    1110
// LE    Less or equal   1111

bool result = false;

switch( condition ) {
   case 0:     // True (not defined)
          result = true;
          break;
   case 1:     // False (not defined)
          break;
   case 2:     // BHI (HI = !C & !Z)
      if ( !( ccr.c | ccr.z) )
          result = true;
      break;
   case 3:     // BLS (LS = C | V)
      if( ccr.c | ccr.z )
          result = true;
      break;
   case 4:     // BCC (CC = !C)
      if ( !ccr.c )
          result = true;
      break;
   case 5:     // BCS (CS = C)
      if( ccr.c )
          result = true;
      break;
   case 6:     // BNE (NE = !Z)
      if ( !ccr.z )
          result = true;
      break;
   case 7:     // BEQ (EQ = Z)
```

```
        if( ccr.z )
            result = true;
        break;
    case 8:       // BVC (VC = !V)
        if( !ccr.v )
            result = true;
        break;
    case 9:       // BVS (VS = V)
        if( ccr.v )
            result = true;
        break;
    case 10: // BPL (PL = !N)
        if( !ccr.n )
            result = true;
        break;
    case 11: // BMI (MI = N)
        if( ccr.n )
            result = true;
        break;
    case 12: // BGE (GE = N & V | !N & !V)
        if( ccr.n & ccr.v | !ccr.n & ccr.v )
            result = true;
        break;
    case 13: // BLT (LT = N & !V | !N & V)
        if( ccr.n & !ccr.v | !ccr.n & ccr.v )
            result = true;
        break;
    case 14: // BGT (GT = N & V & !Z | !N & !V & !Z)
        if( ccr.n & ccr.v & !ccr.z | !ccr.n & !ccr.v & !ccr.z )
            result = true;
        break;
    case 15: // BLE (LE = Z | N & !V | !N & V)
        if( ccr.z | ccr.n & !ccr.v & !ccr.n & ccr.v)
            result = true;
        break;
}

// if false do until
if( result == false ) {
    // dn - 1 -> dn
    d[ reg ].l -= 1;

    // if dn != -1
    if( (signed long int)d[ reg ].l != -1 ) {
        pc = pc + 2 + displ;
    } else {
        pc += 4;
    }

} else {
    pc += 4;
}
```

```
}

void M68008::divsWord()
{
   // size is always word
   // therefore a longword is divided by a word
   // remainder is stored in the upper word
   // quotient in the lower word
   // flags: c is always cleared, x is not affect
   // v if overflow occurs
   // n if negative.. undefined if v is set
   // z if zero occurs, undefined if divide by zero or v is set

   short int sreg,dreg,smode;
   signed long result, address;
   signed long number;
   signed short divider;
   signed short quotient;
   signed long displ;
   sreg = GETBITS( pc, 0x7, 0 );
   smode = GETBITS( pc, 0x38, 3 );
   dreg = GETBITS( pc, 0xE00, 9 );

   // dn / dn -> dn
   if ( smode == 0 ){
      number = ( signed long )d[ dreg ].l;
      divider = ( signed short )d[ sreg ].w;
      if ( divider != 0 )
      {
         result = number % divider;
         result = result << 16;
         quotient = number / divider;
         d[ dreg ].w = quotient;
         d[ dreg ].l += result;
         if ( ( quotient << 16 )> 0 )
         {
            // overflow has occurred
            ccr.v = 1;
         }
         SETNZW( d[ dreg ].w );
      } else {
         // division by zero
         // cause a trap
      }
   }

   // dn / (An) -> dn
   if ( smode == 2 ){
      address = a[ sreg ].l;
      number = ( signed long )d[ dreg].l;
      divider = ( signed short )memory->GetWord( address );
      if ( divider != 0 )
```

```
    {
        result = number % divider;
        result = result << 16;
        quotient = number / divider;
        d[ dreg ].w = quotient;
        d[ dreg].l += result;
        if ( ( quotient << 16 )> 0 )
        {
            // overflow has occurred
            ccr.v = 1;
        }
        SETNZW( d[ dreg ].w );
    } else {
        // division by zero
        // cause a trap
    }
}

// dn / (An)+ -> dn
if ( smode == 3 ){
    address = a[ sreg ].l;
    number = ( signed long )d[ dreg ].l;
    divider = ( signed short )memory->GetWord( address );
    if ( divider != 0 )
    {
        result = number % divider;
        result = result << 16;
        quotient = number / divider;
        d[ dreg ].w = quotient;
        d[ dreg ].l += result;
        if ( ( quotient << 16 )> 0 )
        {
            // overflow has occurred
            ccr.v = 1;
        }
        SETNZW( d[ dreg ].w );
        a[ sreg ].l += 2;
    } else {
        // division by zero
        // cause a trap
    }
}

// dn / -(An) -> dn
if ( smode == 4 ){
    a[ sreg ].l -= 2;
    address = a[ sreg ].l;
    number = ( signed long )d[ dreg ].l;
    divider = ( signed short )memory->GetWord( address );
    if ( divider != 0 )
    {
        result = number % divider;
```

```
            result = result << 16;
            quotient = number / divider;
            d[ dreg ].w = quotient;
            d[ dreg ].l += result;
            if ( ( quotient << 16 )> 0 )
            {
                // overflow has occurred
                ccr.v = 1;
            }
            SETNZW( d[ dreg ].w );
        } else {
            // division by zero
            // cause a trap
        }
    }

    // dn / d16(an)
    if ( smode == 5 )
    {
        displ = (signed short int)memory->GetWord( pc + 2 )+ a[ sreg ].l;
        number = ( signed long )d[ dreg ].l;
        divider = ( signed short )memory->GetWord( displ );
        if ( divider != 0 )
        {
            result = number % divider;
            result = result << 16;
            quotient = number / divider;
            d[ dreg ].w = quotient;
            d[ dreg ].l += result;
            if ( ( quotient << 16 )> 0 )
            {
                // overflow has occurred
                ccr.v = 1;
            }
            SETNZW( d[ dreg ].w );
        } else {
            //division by zero
            // cause a trap
        }
        pc += 2;
    }

    // dn / d8(an, xn)
    if ( smode == 6 )
    {
        displ = (signed char)memory->GetByte( pc + 3 )+ a[ sreg ].l
            + d[ GETBITS( pc + 2, 0xF000, 12 )].l;
        number = ( signed long )d[ dreg ].l;
        divider = ( signed short )memory->GetWord( displ );
        if ( divider != 0 )
        {
            result = number % divider;
```

```
            result = result << 16;
            quotient = number / divider;
            d[ dreg ].w = quotient;
            d[ dreg ].l += result;
            if ( ( quotient << 16 )> 0 )
            {
                // overflow has occurred
                ccr.v = 1;
            }
            SETNZW( d[ dreg ].w );
        } else {
            // division by zero
            // cause a trap
        }
        pc += 2;
    }

    if ( smode == 7 )
    {
        if ( sreg == 0 ){
            // dn / (XXX).W
            displ = memory->GetWord( pc + 2 );
            number = ( signed long )d[ dreg ].l;
            divider = ( signed short )memory->GetLongword( displ );
            if ( divider != 0 )
            {
                result = number % divider;
                result = result << 16;
                quotient = number / divider;
                d[ dreg ].w = quotient;
                d[ dreg ].l += result;
                if ( ( quotient << 16 )> 0 )
                {
                    // overflow has occurred
                    ccr.v = 1;
                }
                SETNZW( d[ dreg ].w );
            } else {
                // division by zero
                // cause a trap
            }
            pc += 2;
        }
        if ( sreg == 1 ){
            // dn / (XXX).l
            displ = memory->GetLongword( pc + 2 );
            number = ( signed long )d[ dreg ].l;
            divider = ( signed short )memory->GetLongword( displ );
            if ( divider != 0 )
            {
                result = number % divider;
                result = result << 16;
```

```
            quotient = number / divider;
            d[ dreg ].w = quotient;
            d[ dreg ].l += result;
            if ( ( quotient << 16 )> 0 )
            {
               // overflow has occurred
               ccr.v = 1;
            }
            SETNZW( d[ dreg ].w );
         } else {
            // division by zero
            // cause a trap
         }
         pc += 4;
      }

      if ( sreg == 4 )
      {
         // dn / #<data>
         displ = memory->GetWord( pc + 2 );
         number = ( signed long )d[ dreg ].l;
         divider = ( signed short )displ;
         if ( divider != 0 )
         {
            result = number % divider;
            result = result << 16;
            quotient = number / divider;
            d[ dreg ].w = quotient;
            d[ dreg ].l += result;
            if ( ( quotient << 16 )> 0 )
            {
               // overflow has occurred
               ccr.v = 1;
            }
            SETNZW( d[ dreg ].w );
         } else {
            // division by zero
            // cause a trap
         }
         pc += 2;
      }
   }

   ccr.c = 0;
   pc += 2;
}

void M68008::divsLong()
{
   // invalid size code
}
```

```
void M68008::divuWord()
{
    short int sreg,dreg,smode;
    unsigned long result, address;
    unsigned long number;
    unsigned short divider;
    unsigned short quotient;
    unsigned long displ;
    sreg = GETBITS( pc, 0x7, 0 );
    smode = GETBITS( pc, 0x38, 3 );
    dreg = GETBITS( pc, 0xE00, 9 );

    // dn / dn -> dn
    if ( smode == 0 ){
        number = ( unsigned long )d[ dreg ].l;
        divider = ( unsigned short )d[ sreg ].w;
        if ( divider != 0 )
        {
            result = number % divider;
            result = result << 16;
            quotient = number / divider;
            d[ dreg ].w = quotient;
            d[ dreg ].l += result;
            if ( ( quotient << 16 )> 0 )
            {
                // overflow has occurred
                ccr.v = 1;
            }
            SETNZW( d[ dreg ].w );
        } else {
            // division by zero
            // cause a trap
        }
    }

    // dn / (An) -> dn
    if ( smode == 2 ){
        address = a[ sreg ].l;
        number = ( unsigned long )d[ dreg].l;
        divider = ( unsigned short )memory->GetWord( address );
        if ( divider != 0 )
        {
            result = number % divider;
            result = result << 16;
            quotient = number / divider;
            d[ dreg ].w = quotient;
            d[ dreg].l += result;
            if ( ( quotient << 16 )> 0 )
            {
                // overflow has occurred
                ccr.v = 1;
            }
```

```
        SETNZW( d[ dreg ].w );
    } else {
        // division by zero
        // cause a trap
    }
}

// dn / (An)+ -> dn
if ( smode == 3 ){
    address = a[ sreg ].l;
    number = ( unsigned long )d[ dreg ].l;
    divider = ( unsigned short )memory->GetWord( address );
    if ( divider != 0 )
    {
        result = number % divider;
        result = result << 16;
        quotient = number / divider;
        d[ dreg ].w = quotient;
        d[ dreg ].l += result;
        if ( ( quotient << 16 )> 0 )
        {
            // overflow has occurred
            ccr.v = 1;
        }
        SETNZW( d[ dreg ].w );
        a[ sreg ].l += 2;
    } else {
        // division by zero
        // cause a trap
    }
}

// dn / -(An) -> dn
if ( smode == 4 ){
    a[ sreg ].l -= 2;
    address = a[ sreg ].l;
    number = ( unsigned long )d[ dreg ].l;
    divider = ( unsigned short )memory->GetWord( address );
    if ( divider != 0 )
    {
        result = number % divider;
        result = result << 16;
        quotient = number / divider;
        d[ dreg ].w = quotient;
        d[ dreg ].l += result;
        if ( ( quotient << 16 )> 0 )
        {
            // overflow has occurred
            ccr.v = 1;
        }
        SETNZW( d[ dreg ].w );
    } else {
```

```
        // division by zero
        // cause a trap
    }
}

// dn / d16(an)
if ( smode == 5 )
{
    displ = ( unsigned short int)memory->GetWord( pc + 2 )+ a[ sreg ].l;
    number = ( unsigned long )d[ dreg ].l;
    divider = ( unsigned short )memory->GetWord( displ );
    if ( divider != 0 )
    {
        result = number % divider;
        result = result << 16;
        quotient = number / divider;
        d[ dreg ].w = quotient;
        d[ dreg ].l += result;
        if ( ( quotient << 16 )> 0 )
        {
            // overflow has occurred
            ccr.v = 1;
        }
        SETNZW( d[ dreg ].w );
    } else {
        //division by zero
        // cause a trap
    }
    pc += 2;
}

// dn / d8(an, xn)
if ( smode == 6 )
{
    displ = ( unsigned char)memory->GetByte( pc + 3 )+ a[ sreg ].l
        + d[ GETBITS( pc + 2, 0xF000, 12 )].l;
    number = ( unsigned long )d[ dreg ].l;
    divider = ( unsigned short )memory->GetWord( displ );
    if ( divider != 0 )
    {
        result = number % divider;
        result = result << 16;
        quotient = number / divider;
        d[ dreg ].w = quotient;
        d[ dreg ].l += result;
        if ( ( quotient << 16 )> 0 )
        {
            // overflow has occurred
            ccr.v = 1;
        }
        SETNZW( d[ dreg ].w );
    } else {
```

```
        // division by zero
        // cause a trap
    }
    pc += 2;
}

if ( smode == 7 )
{
    if ( sreg == 0 ){
        // dn / (XXX).W
        displ = memory->GetWord( pc + 2 );
        number = ( unsigned long )d[ dreg ].l;
        divider = ( unsigned short )memory->GetLongword( displ );
        if ( divider != 0 )
        {
            result = number % divider;
            result = result << 16;
            quotient = number / divider;
            d[ dreg ].w = quotient;
            d[ dreg ].l += result;
            if ( ( quotient << 16 )> 0 )
            {
                // overflow has occurred
                ccr.v = 1;
            }
            SETNZW( d[ dreg ].w );
        } else {
            // division by zero
            // cause a trap
        }
        pc += 2;
    }
    if ( sreg == 1 ){
        // dn / (XXX).l
        displ = memory->GetLongword( pc + 2 );
        number = ( unsigned long )d[ dreg ].l;
        divider = ( unsigned short )memory->GetLongword( displ );
        if ( divider != 0 )
        {
            result = number % divider;
            result = result << 16;
            quotient = number / divider;
            d[ dreg ].w = quotient;
            d[ dreg ].l += result;
            if ( ( quotient << 16 )> 0 )
            {
                // overflow has occurred
                ccr.v = 1;
            }
            SETNZW( d[ dreg ].w );
        } else {
            // division by zero
```

```
            // cause a trap
        }
        pc += 4;
    }

    if ( sreg == 4 )
    {
        // dn / #<data>
        displ = memory->GetWord( pc + 2 );
        number = ( unsigned long )d[ dreg ].l;
        divider = ( unsigned short )displ;
        if ( divider != 0 )
        {
            result = number % divider;
            result = result << 16;
            quotient = number / divider;
            d[ dreg ].w = quotient;
            d[ dreg ].l += result;
            if ( ( quotient << 16 )> 0 )
            {
                // overflow has occurred
                ccr.v = 1;
            }
            SETNZW( d[ dreg ].w );
        } else {
            // division by zero
            // cause a trap
        }
        pc += 2;
    }
}

ccr.c = 0;
pc += 2;

}

void M68008::divuLong()
{
    //invalid size code
}

void M68008::eor()
{

    int reg, opmode, mode, eaReg;
    DATA_REGISTER( result );
    signed long displ;


    reg    = GETBITS( pc, 0x0E00, 9 );
    opmode = GETBITS( pc, 0x01C0, 6 );
```

```
mode  = GETBITS( pc, 0x0038, 3 );
eaReg = GETBITS( pc, 0x0007, 0 );

// dn,Dn
if( mode == 0 )
{
    // byte
    if( opmode == 4 )
    {
        result.b = d[ eaReg ].b ^ d[ reg ].b;
        SETNZ( result.b );
        ccr.v = ccr.c = 0;
        d[ reg ].b = result.b;
    }

    // word
    if( opmode == 5 )
    {
        result.w = d[ eaReg ].w ^ d[ reg ].w;
        SETNZ( result.w );
        ccr.v = ccr.c = 0;
        d[ reg ].w = result.w;
    }

    // long
    if( opmode == 6 )
    {
        result.l = d[ eaReg ].l ^ d[ reg ].l;
        SETNZ( result.l );
        ccr.v = ccr.c = 0;
        d[ reg ].l = result.l;
    }

    pc += 2;
}

// (an),dn
if( mode == 2 )
{
    // byte
    if( opmode == 4 )
    {
        result.b = memory->GetByte( a[ eaReg ].l )^ d[ reg ].b;
        SETNZ( result.b );
        ccr.v = ccr.c = 0;
        memory->SetByte( result.b, a[ eaReg ].l );
    }

    // word
    if( opmode == 5 )
    {
        result.w = memory->GetWord( a[ eaReg ].l )^ d[ reg ].w;
```

```
        SETNZ( result.w );
        ccr.v = ccr.c = 0;
        memory->SetWord( result.w, a[ eaReg ].l );
    }

    // long
    if( opmode == 6 )
    {
        result.l = memory->GetLongword( a[ eaReg ].l )^ d[ reg ].l;
        SETNZ( result.l );
        ccr.v = ccr.c = 0;
        memory->SetLongword( result.l, a[ eaReg ].l );
    }

    pc += 2;
}

// (an)+,dn
if( mode == 3 )
{
    // byte
    if( opmode == 4 )
    {
        result.b = memory->GetByte( a[ eaReg ].l )^ d[ reg ].b;
        SETNZ( result.b );
        ccr.v = ccr.c = 0;
        memory->SetByte( result.b, a[ eaReg ].l );
        a[ eaReg ].l += 1;
    }

    // word
    if( opmode == 5 )
    {
        result.w = memory->GetWord( a[ eaReg ].l )^ d[ reg ].w;
        SETNZ( result.w );
        ccr.v = ccr.c = 0;
        memory->SetWord( result.w, a[ eaReg ].l );
        a[ eaReg ].l += 2;
    }

    // long
    if( opmode == 6 )
    {
        result.l = memory->GetLongword( a[ eaReg ].l )^ d[ reg ].l;
        SETNZ( result.l );
        ccr.v = ccr.c = 0;
        memory->SetLongword( result.l, a[ eaReg ].l );
        a[ eaReg ].l += 4;
    }

    pc += 2;
}
```

```
// -(an),dn
if( mode== 4 )
{

    // byte
    if( opmode == 4 )
    {
        a[ eaReg ].l -= 1;
        result.b = memory->GetByte( a[ eaReg ].l )^ d[ reg ].b;
        SETNZ( result.b );
        ccr.v = ccr.c = 0;
        memory->SetByte( result.b, a[ eaReg ].l );
    }

    // word
    if( opmode == 5 )
    {
        a[ eaReg ].l -= 2;
        result.w = memory->GetWord( a[ eaReg ].l )^ d[ reg ].w;
        SETNZ( result.w );
        ccr.v = ccr.c = 0;
        memory->SetWord( result.w, a[ eaReg ].l );
    }

    // long
    if( opmode == 6 )
    {
        a[ eaReg ].l -= 4;
        result.l = memory->GetLongword( a[ eaReg ].l )^ d[ reg ].l;
        SETNZ( result.l );
        ccr.v = ccr.c = 0;
        memory->SetLongword( result.l, a[ eaReg ].l );
    }

    pc += 2;
}

// d16(an),dn
if( mode == 5 )
{
    displ = (signed short int)memory->GetWord( pc + 2 )+ a[ eaReg ].l;

    // byte
    if( opmode == 4 )
    {
        result.b = memory->GetByte( displ )^ d[ reg ].b;
        SETNZ( result.b );
        ccr.v = ccr.c = 0;
        memory->SetByte( result.b, displ );
    }
```

```
    // word
    if ( opmode == 5 )
    {
        result.w = memory->GetWord( displ )^ d[ reg ].w;
        SETNZ( result.w );
        ccr.v = ccr.c = 0;
        memory->SetWord( result.w, displ );
    }

    // long
    if ( opmode == 6 )
    {
        result.l = memory->GetLongword( displ )^ d[ reg ].l;
        SETNZ( result.l );
        ccr.v = ccr.c = 0;
        memory->SetLongword( result.l, displ );
    }

    pc += 4;
}

// d8(an,xn),dn
if ( mode == 6 )
{
    displ = (signed char)memory->GetByte( pc + 3 )+ a[ eaReg ].l
        + d[ GETBITS( pc + 2, 0xF000, 12 )].l;

    // byte
    if ( opmode == 4 )
    {
        result.b = memory->GetByte( displ )^ d[ reg ].b;
        SETNZ( result.b );
        ccr.v = ccr.c = 0;
        memory->SetByte( result.b, displ );
    }

    // word
    if ( opmode == 5 )
    {
        result.w = memory->GetWord( displ )^ d[ reg ].w;
        SETNZ( result.w );
        ccr.v = ccr.c = 0;
        memory->SetWord( result.w, displ );
    }

    // long
    if ( opmode == 6 )
    {
        result.l = memory->GetLongword( displ )^ d[ reg ].l;
        SETNZ( result.l );
        ccr.v = ccr.c = 0;
        memory->SetLongword( result.l, displ );
```

```
    }

    pc += 4;
}

if( mode == 7)
{
    // (xxx).w,dn
    if( eaReg == 0 )
    {
        displ = memory->GetWord( pc + 2 );

        // byte
        if( opmode == 4 )
        {
            result.b = memory->GetByte( displ )^ d[ reg ].b;
            SETNZ( result.b );
            ccr.v = ccr.c = 0;
            memory->SetByte( result.b, displ );
        }

        // word
        if( opmode == 5 )
        {
            result.w = memory->GetWord( displ )^ d[ reg ].w;
            SETNZ( result.w );
            ccr.v = ccr.c = 0;
            memory->SetWord( result.w, displ );
        }

        // long
        if( opmode == 6 )
        {
            result.l = memory->GetLongword( displ )^ d[ reg ].l;
            SETNZ( result.l );
            ccr.v = ccr.c = 0;
            memory->SetLongword( result.l, displ );
        }
        pc += 4;
    }

    // (xxx).l,dn
    if( eaReg == 1 )
    {
        displ = memory->GetLongword( pc + 2 );

        // byte
        if( opmode == 4 )
        {
            result.b = memory->GetByte( displ )^ d[ reg ].b;
            SETNZ( result.b );
            ccr.v = ccr.c = 0;
```

```
            memory->SetByte( result.b, displ );
        }

        // word
        if( opmode == 5 )
        {
            result.w = memory->GetWord( displ )^ d[ reg ].w;
            SETNZ( result.w );
            ccr.v = ccr.c = 0;
            memory->SetWord( result.w, displ );
        }

        // long
        if( opmode == 6 )
        {
            result.l = memory->GetLongword( displ )^ d[ reg ].l;
            SETNZ( result.l );
            ccr.v = ccr.c = 0;
            memory->SetLongword( result.l, displ );
        }
        pc += 6;
    }
  }

  // OMITTED
  // #data,dn (see ORI)
  // d16(PC),dn
  // d8(PC,xn)
}

void M68008::eori()
{
  long int size, mode, reg;
  unsigned long displ;
  DATA_REGISTER( imm );
  DATA_REGISTER( result );


  size = GETBITS( pc, 0x00C0, 6 );
  mode = GETBITS( pc, 0x0038, 3 );
  reg  = GETBITS( pc, 0x0007, 0 );

  // Fetch the immediate value
  if( size == 0 )
    imm.b = memory->GetByte( pc + 3 );
  if( size == 1 )
    imm.w = memory->GetWord( pc + 2 );
  if( size == 2 )
    imm.l = memory->GetLongword( pc + 2 );

  // #data,dn
  if( mode == 0 )
```

```
{
    if( size == 0 )
    {
        result.b = imm.b ^ d[ reg ].b;
        d[ reg ].b = result.b;
        SETNZ( d[ reg ].b );
        ccr.c = ccr.v = 0;
        pc += 4;
    }
    if( size == 1 )
    {
        result.w = imm.w ^ d[ reg ].w;
        d[ reg ].w = result.w;
        SETNZW( d[ reg ].w );
        ccr.c = ccr.v = 0;
        pc += 4;
    }
    if( size == 2 )
    {
        result.l = imm.l ^ d[ reg ].l;
        d[ reg ].l = result.l;
        SETNZL( d[ reg ].l );
        ccr.c = ccr.v = 0;
        pc += 6;
    }
}

// #data,(an)
if( mode == 2 )
{
    if( size == 0 )
    {
        result.b = imm.b ^ memory->GetByte( a[ reg ].l );
        memory->SetByte( result.b, a[ reg ].l );
        SETNZ( memory->GetByte( a[ reg ].l ));
        ccr.c = ccr.v = 0;
        pc += 4;
    }
    if( size == 1 )
    {
        result.w = imm.w ^ memory->GetWord( a[ reg ].l );
        memory->SetWord( result.w, a[ reg ].l );
        SETNZ( memory->GetWord( a[ reg ].l ));
        ccr.c = ccr.v = 0;
        pc += 4;
    }
    if( size == 2 )
    {
        result.l = imm.l ^ memory->GetLongword( a[ reg ].l );
        memory->SetLongword( result.l, a[ reg ].l );
        SETNZ( memory->GetLongword( a[ reg ].l ));
        ccr.c = ccr.v = 0;
```

```
        pc += 6;
    }
}

// #data,(an)+
if( mode == 3 )
{
    if( size == 0 )
    {
        result.b = imm.b ^ memory->GetByte( a[ reg ].l );
        memory->SetByte( result.b, a[ reg ].l );
        SETNZ( memory->GetByte( a[ reg ].l ));
        ccr.c = ccr.v = 0;
        pc += 4;
        a[ reg ].l += 1;
    }
    if( size == 1 )
    {
        result.w = imm.w ^ memory->GetWord( a[ reg ].l );
        memory->SetWord( result.w, a[ reg ].l );
        SETNZ( memory->GetWord( a[ reg ].l ));
        ccr.c = ccr.v = 0;
    pc += 4;
        a[ reg ].l += 2;
    }
    if( size == 2 )
    {
        result.l = imm.l ^ memory->GetLongword( a[ reg ].l );
        memory->SetLongword( result.l, a[ reg ].l );
        SETNZ( memory->GetLongword( a[ reg ].l ));
        ccr.c = ccr.v = 0;
        pc += 6;
        a[ reg ].l += 4;
    }
}

// #data,-(an)
if( mode == 4 )
{
    if( size == 0 )
    {
        a[ reg ].l -= 1;
        result.b = imm.b ^ memory->GetByte( a[ reg ].l );
        memory->SetByte( result.b, a[ reg ].l );
        SETNZ( memory->GetByte( a[ reg ].l ));
        ccr.c = ccr.v = 0;
        pc += 4;
    }
    if( size == 1 )
    {
        a[ reg ].l -= 2;
```

```
        result.w = imm.w ^ memory->GetWord( a[ reg ].l );
        memory->SetWord( result.w, a[ reg ].l );
        SETNZ( memory->GetWord( a[ reg ].l ));
        ccr.c = ccr.v = 0;
        pc += 4;
    }
    if( size == 2 )
    {
        a[ reg ].l -= 4;
        result.l = imm.l ^ memory->GetLongword( a[ reg ].l );
        memory->SetLongword( result.l, a[ reg ].l );
        SETNZ( memory->GetLongword( a[ reg ].l ));
        ccr.c = ccr.v = 0;
        pc += 6;
    }
}

// #data,d16(an)
if( mode == 5 )
{
    if( size == 0 )
    {
        displ = (signed short int)memory->GetWord( pc + 4 )+ a[ reg ].l;
        result.b = imm.b ^ memory->GetByte( displ );
        memory->SetByte( result.b, displ );
        SETNZ( memory->GetByte( displ ));
        ccr.c = ccr.v = 0;
        pc += 6;
    }
    if( size == 1 )
    {
        displ = (signed short int)memory->GetWord( pc + 4 )+ a[ reg ].l;
        result.w = imm.w ^ memory->GetWord( displ );
        memory->SetWord( result.w, displ );
        SETNZ( memory->GetWord( displ ));
        ccr.c = ccr.v = 0;
        pc += 6;
    }
    if( size == 2 )
    {
        displ = (signed short int)memory->GetWord( pc + 6 )+ a[ reg ].l;
        result.l = imm.l ^ memory->GetLongword( displ );
        memory->SetLongword( result.l, displ );
        SETNZ( memory->GetLongword( displ ));
        ccr.c = ccr.v = 0;
        pc += 8;
    }
}

// #data,d8(an,xn)
if( mode == 6 )
{
```

```
    if( size == 0 )
    {
        displ = (signed char)memory->GetByte( pc + 5 )+ a[ reg ].l
            + d[ GETBITS( pc + 4, 0xF000, 12 )].l;
        result.b = imm.b ^ memory->GetByte( displ );
        memory->SetByte( result.b, displ );
        SETNZ( memory->GetByte( displ ));
        ccr.c = ccr.v = 0;
        pc += 6;
    }
    if( size == 1 )
    {
        displ = (signed char)memory->GetByte( pc + 5 )+ a[ reg ].l
            + d[ GETBITS( pc + 4, 0xF000, 12 )].l;
        result.w = imm.w ^ memory->GetWord( displ );
        memory->SetWord( result.w, displ );
        SETNZ( memory->GetWord( displ ));
        ccr.c = ccr.v = 0;
        pc += 6;
    }
    if( size == 2 )
    {
        displ = (signed char)memory->GetByte( pc + 7 )+ a[ reg ].l
            + d[ GETBITS( pc + 6, 0xF000, 12 )].l;
        result.l = imm.l ^ memory->GetLongword( displ );
        memory->SetLongword( result.l, displ );
        SETNZ( memory->GetLongword( displ ));
        ccr.c = ccr.v = 0;
        pc += 8;
    }
}

if( mode == 7 )
{
    // #data,(xxx).w
    if( reg == 0 )
    {
        if( size == 0 )
        {
            displ = memory->GetWord( pc + 4 );
            result.b = imm.b ^ memory->GetByte( displ );
            memory->SetByte( result.b, displ );
            SETNZ( memory->GetByte( displ ));
            ccr.c = ccr.v = 0;
            pc += 6;
        }
        if( size == 1 )
        {
            displ = memory->GetWord( pc + 4 );
            result.w = imm.w ^ memory->GetWord( displ );
            memory->SetWord( result.w, displ );
            SETNZ( memory->GetWord( displ ));
```

```
            ccr.c = ccr.v = 0;
            pc += 6;
        }
        if( size == 2 )
        {
            displ = memory->GetWord( pc + 6 );
            result.l = imm.l ^ memory->GetLongword( displ );
            memory->SetLongword( result.l, displ );
            SETNZ( memory->GetLongword( displ ));
            ccr.c = ccr.v = 0;
            pc += 8;
        }
    }

    // #data,(xxx).l
    if( reg == 1 )
    {
        if( size == 0 )
        {
            displ = memory->GetLongword( pc + 4 );
            result.b = imm.b ^ memory->GetByte( displ );
            memory->SetByte( result.b, displ );
            SETNZ( memory->GetByte( displ ));
            ccr.c = ccr.v = 0;
        pc += 8;
        }
        if( size == 1 )
        {
            displ = memory->GetLongword( pc + 4 );
            result.w = imm.w ^ memory->GetWord( displ );
            memory->SetWord( result.w, displ );
            SETNZ( memory->GetWord( displ ));
            ccr.c = ccr.v = 0;
            pc += 8;
        }
        if( size == 2 )
        {
            displ = memory->GetLongword( pc + 6 );
            result.l = imm.l ^ memory->GetLongword( displ );
            memory->SetLongword( result.l, displ );
            SETNZ( memory->GetLongword( displ ));
            ccr.c = ccr.v = 0;
            pc += 10;
        }
    }
  }
}

void M68008::eoriCcr()
{

    if ( GETBITS( pc + 2, 0x0010, 4 )){
```

```
        ccr.x = !ccr.x;
    }

    if (GETBITS( pc + 2, 0x0008, 3 )){
        ccr.n = !ccr.n;
    }

    if (GETBITS( pc + 2, 0x0004, 2 )){
        ccr.z = !ccr.z;
    }

    if (GETBITS( pc + 2, 0x0002, 1 )){
        ccr.v = !ccr.v;
    }
    if (GETBITS( pc + 2, 0x0001, 0 )){
        ccr.c = !ccr.c;
    }
    pc += 4;
}

void M68008::exg()
{
    // size is always a longword

    long temp;
    short int opmode, regx, regy;
    regy = GETBITS( pc, 0x7, 0 );
    opmode = GETBITS( pc, 0xF8, 3 );
    regx = GETBITS( pc, 0xE00, 9 );


    // data registers
    if ( opmode == 8 ){
        temp = d[ regy ].l;
        d[ regy ].l = d[ regx ].l;
        d[ regx ].l = temp;
    }

    // address registers
    if ( opmode == 9 ){
        temp = a[ regy ].l;
        a[ regy ].l = a[ regx ].l;
        a[ regx ].l = temp;
    }

    // data and address registers
    // regy is the address register and
    // regx is the data register
    if ( opmode == 17 ){
        temp = a[ regy ].l;
        a[ regy ].l = d[ regx ].l;
        d[ regx ].l = temp;
```

```
    }
    pc += 2;
    // flags are not affected
}

void M68008::ext()
{
    short int opmode, reg;
    opmode = GETBITS( pc, 0x1C0, 6 );
    reg = GETBITS( pc, 0x7, 0 );
    DATA_REGISTER( result );

    if ( opmode == 2 ){
        // byte to word operation
        result.b = d[ reg ].b;
        result.w = (signed char)result.b;
        SETNZW( result.w );
        d[ reg ].w = result.w;
    }

    if ( opmode == 3 ){
        // word to longword operation
        result.w = d[ reg ].w;
        result.l = (signed short)result.w;
        SETNZL( result.l );
        d[ reg ].l = result.l;
    }
    pc += 2;
}

void M68008::illegal()
{

    unsigned short int status = 0;
    // pushs the pc and ccr to the stack then updates PC with 0x10

    a[ 7 ].l -= 4;
    memory->SetLongword( pc, a[ 7 ].l );

    status = status | ccr.c;
    status = status | ( ccr.v << 1);
    status = status | ( ccr.z << 2);
    status = status | ( ccr.n << 3);
    status = status | ( ccr.x << 4);

    a[ 7 ].l -= 2;
    memory->SetWord( status, a[ 7 ].l );

    PUSH( "Return Address", 4 );
    PUSH( "CCR", 2 );

    pc = memory->GetLongword( 0x10 );
```

```
}

void M68008::jmp() {

    long newpc = pc;
    //displacement = pc + 2 + operand
    //new pc = pc + ( pc + 2 + operand )

    int ch = memory->GetWord( pc );

    // if the mode is 111
    if ( GETBITS( pc, 56, 0 )== 56 ){

        // and the register is 000
        // jmp (xxx).w
        if ( GETBITS( pc, 7, 0 )== 0 ){
            printf( "jmp: word value\n" );
            newpc = memory->GetWord( pc + 2 );
        }

        // and the register is 001
        // jmp (www).l
        if ( GETBITS( pc, 7, 0 )== 1 ){
            printf( "jmp: long value\n" );
            newpc = memory->GetLongword( pc + 2 );
        }

        /*
        // and if the register is 010
        // jmp (d16,PC)
        // OMIT : This instruction is not currently implemented.
        if ( GETBITS( pc, 7, 0 )== 2 ){
            printf( "jmp: word value displacement to PC\n" );

            int operand = memory->GetWord( pc + 2 );

            int displacement = (signed short int)memory->GetWord( pc )+ pc;

            newpc += displacement;
        }
        */


    }

    // if the mode is 010
    // jmp (An)
    if ( GETBITS( pc, 56, 0 )== 16 ){

        // we need to decode the register number
        int var1;
```

```
        var1 = ch & 7;

        if (var1 >= 8) {
            printf("ERROR!");
        } else {
            var1 = a[var1].l;
            newpc = memory->GetLongword( var1 );
            printf( "jmp: address register\n" );
        }
    }

    // if the mode is 101
    // jmp (d16, An)
    if ( GETBITS( pc, 56, 0 )== 40 ){

        a[0].l = 0x110;

        printf( "jmp: word displacement to address register\n" );

        // we need to decode the register number
        int var1;

        var1 = ch & 7;

        if (var1 >= 8) {

            printf("ERROR!");

        } else {
            int displacement = (signed short int)memory->GetWord( pc + 2 )+ a[ var1 ].l;
            newpc = displacement;
        }

    }

    // OMIT: modes 110 and last two 111 instructions are not implemented
    // compiler wouldn't recognise these instructions
    // come back to it later

    pc = newpc;

}

void M68008::jsr() {
    /* The function of the jsr instruction is to preserve the long value of
    the address for the next instruction to the stack and jump to that location.
        PC + instruction length -> Stack
        Stack pointer is updated.l
        Destination address -> PC */

    long newpc = pc;
```

```
// if the mode is 111
if ( GETBITS( pc, 56, 0 )== 56 ){

   // if the register is 000 then the instruction type is
   // jsr (xxx).w
   if ( GETBITS( pc, 7, 0) == 0 ){

      printf( "jsr: with a word value.\n" );

      // before incrementing the pc to the next instruction
      // we must get the destination address
      long dest = memory->GetWord( pc + 2 );

      // increment the PC to the next instruction.
      newpc += 4;

      // Add the long word PC to the stack.
      a[7].l -= 4;                    // first decrement pointer so its pointing at the next space
      memory->SetLongword( newpc, a[7].l );  // set longword on stack to be pc

      // Stack operations.
      PUSH( "Return address", 4);

      // destination address -> pc
      newpc = dest;

   }

   // if the register is 1 then the instruction type is
   // jsr (xxx).l
   if ( GETBITS( pc, 7, 0) == 1 ){

      printf( "jsr: with a long word value.\n" );

      // before incrementing the pc to the next instruction
      // we must get the destination address
      long dest = memory->GetLongword( pc + 2 );

      // increment the PC to the next instruction.
      newpc += 6;

      // Add the long word PC to the stack.
      a[7].l -= 4;                    // first decrement pointer so its pointing at the next space
      memory->SetLongword( newpc, a[7].l );  // set longword on stack to be pc

      // Stack operations
      PUSH( "Return address", 4);

      // destination address -> pc
      newpc = dest;
   }
```

```
      // if the register is 010 then the instruction type is
      // jsr (d16, PC)
      if ( GETBITS( pc, 7, 0) == 2 ){
/*
         printf( "jsr: longword displacement to PC\n" );

         //  before incrementing the pc to the next instruction
         // we get the destination address
         int displacement = memory->GetWord( pc + 2 );

         if ( displacement >= 32509 && displacement <= 65276) {
            // displacement is negative
            // so we need to calculate the correct displacement in integer
            // so
            displacement -= 65272;
         } else {
            // displacement is positive
            // so we need to calculate the correct displacement in integer
            // so
            displacement -= 65276;
         }

         long dest = displacement + pc;

         // increment the PC to the next instruction
         newpc += 4;

         // Add the long word PC to the stack.
         a[7].l -= 4;                    // first decrement pointer so its pointing at the next space
         memory->SetLongword( newpc, a[7].l );  // set longword on stack to be pc

         // destination address -> pc
         newpc = dest;
*/
         newpc += 4;

      }

   }

   // if the mode is 010
   if ( GETBITS( pc, 56, 0 )== 16 ){

      // then the instruction type is
      // jsr (An)
      // we need to decode the register number
      int var1;

      var1 = memory->GetWord( pc )& 7;

      if (var1 >= 8) {
         printf("ERROR!");
```

```
    } else {

        printf( " jsr : address pointed to by an address register.\n" );
        // get the address location of the value pointed to by the adress register
        long dest = a[var1].l;
        dest = memory->GetLongword( dest );

        // increment the PC to the next instruction
        newpc += 2;

        // Add the long word PC to the stack.
        a[7].l -= 4;                    // first decrement pointer so its pointing at the next space
        memory->SetLongword( newpc, a[7].l );  // set longword on stack to be pc

        // Stack operations
        PUSH( "Return address", 4 );

        // destination address -> pc
        newpc = dest;

    }
  }

  // if the mode is d16(an)
  if ( GETBITS( pc, 56, 0 )== 40 ){
/*
    a[0].l = 0x106;

    // then the instruction type is
    // jsr (d16,An)
    // we need to decode the register number

    printf( "jmp: word displacement to address register\n" );

    // we need to decode the register number
    int var1;

    var1 = memory->GetWord( pc )& 7;



    if (var1 >= 8) {

        printf( "ERROR!" );

    } else {

        // get the address location of the value pointed to by the adress register
        long dest = a[var1].l;
        dest = memory->GetLongword( dest );
        int displacement = memory->GetWord( pc + 2 );
```

```
            if ( displacement >= 32768) {
               // displacement is negative
               // so we need to calculate the correct displacement in integer
               // so
               displacement -= 65536;
            }

            dest += displacement;

            // increment the PC to the next instruction
            newpc += 4;

            // Add the long word PC to the stack.
            a[7].l -= 4;                    // first decrement pointer so its pointing at the next space
            memory->SetLongword( newpc, a[7].l );  // set longword on stack to be pc

            // destination address -> pc
            newpc = dest;

         }
*/

         // if mode is d8(An,Xn)
         // !Ommitted
         newpc += 4;
      }

      pc = newpc;
}

void M68008::lea() {

   long int reg, mode, eareg;
   unsigned long disp;
   reg = GETBITS( pc, 0xe00, 9 );
   mode = GETBITS( pc, 0x38, 3 );
   eareg = GETBITS( pc, 0x7, 0 );

   // (An)
   if( mode == 2) {

      a[ reg ].l = memory->GetLongword( a[ eareg ].l );
      pc += 2;
   }

   // d16,An
   if ( mode == 5) {
      disp = (signed short int)memory->GetWord( pc + 2 )+ a[ eareg ].l;
      a[ reg ].l = memory->GetLongword( disp );
      pc += 4;
   }
```

```
    // d8(An,Xn)
    if( mode == 6) {
        disp = (signed char)memory->GetByte( pc + 3 )+ a[ eareg ].l
            + d[ GETBITS( pc + 2, 0xF000, 12 )].l;

        a[ reg ]. l = memory->GetLongword( disp );
        pc += 4;
    }

    // No thing I can say in here
    if ( mode == 7) {

        // (XXX).W
        if( eareg == 0) {
            a[ reg ]. l = memory->GetWord( pc + 2 );
            pc += 4;
        }

        // (XXX).L
        if( eareg == 1) {
            a[ reg ]. l = memory->GetLongword( pc + 2 );
            pc += 6;
        }

        /* OMIT:


        // d16(PC)
        if( eareg == 2) {
            a[ reg ]. l = memory->GetLongword( (signed long int)memory->GetWord( pc + 2 )+ pc);
            pc += 4;
        }

        // d8(PC,Xn)
        if( eareg == 3) {
        }
        */
    }

}

void M68008::linkWord() {
    signed short int displ = memory->GetWord( pc + 2 );
    short int reg = GETBITS( pc, 0x0007, 0 );
    char *buf = new char[ 10 ];

    // makes room on the stack.
    a [ 7 ]. l -= 4;

    // pushes the specified register to the stack.
    memory->SetLongword( a[ reg ].l, a[ 7 ].l );
```

```
    sprintf ( buf, "a%d (by LINK)", reg );
    PUSH( buf, 4 );

    // Loads the register with the SP
    a[ reg ].l = a [ 7 ].l;

    // increments the SP with the displacement.
    a [ 7 ].l += displ;

    pc += 4;
}

/* ! Not supported by the 68008 chipset. */
void M68008::linkLong() {
    pc += 4;
}

void M68008::lsl_rReg() {


    short int reg = GETBITS( pc, 0x0007, 0 );
    short int ir = GETBITS( pc, 0x0020, 5);
    short int size = GETBITS( pc, 0x00c0, 6 );
    short int dr = GETBITS( pc, 0x0100, 8 );
    short int creg = GETBITS( pc, 0x0e00, 9 );
    long int contents = 0;

    short int count;

    if( ir == 0 ){

        // gets the count
        if( creg == 0 ){
            count = 8;
        } else {
            count = creg;
        }

    } else {
        creg = d[ creg ].l;
    }

    // if the operation is a byte.
    if( size == 0 ){
        // This is the section where we get the information necessary.
        contents = d[reg].b;
        int last;

        // does the operation
        for( int i = 0; i < count; i++ ){

            // if it is shifting right
```

```
        if( dr == 0) {
            last = contents & 0x0001;
            contents = contents >> 1;
        } else {
            // or shifting left.
            last = (contents & 0x0080) >> 7;
            contents = contents << 1;
        }
    }

    // sets the other flags
    ccr.x = ccr.c = last;
    contents = contents & 0x00ff;
    ccr.v = 0;
    ccr.z = (contents == 0);
    ccr.n = (contents >> 7) & 0x0001;


    // assigns the result back.
    d[reg].b = contents;

}

if( size == 1 ){
    contents = d[reg].w;
    int last;

    // does the operation
    for( int i = 0; i < count; i++ ){

        // if it is shifting right
        if( dr == 0) {
            last = contents & 0x0001;
            contents = contents >> 1;
        } else {
            // or shifting left.
            last = (contents & 0x8000) >> 15;
            contents = contents << 1;
        }
    }

    // sets the other flags
    ccr.x = ccr.c = last;
    contents = contents & 0xffff;
    ccr.v = 0;
    ccr.z = (contents == 0);
    ccr.n = (contents >> 15) & 0x0001;

    // This section stores the information back to the register again.
    d[reg].w = contents;
}
```

```
    if( size == 2 ){
        contents = d[reg].l;
        int last;

        // does the operation
        for( int i = 0; i < count; i++ ){

            // if it is shifting right
            if( dr == 0) {
                last = contents & 0x0001;
                contents = contents >> 1;
            } else {
                // or shifting left.
                last = (contents & 0x80000000) >> 31;
                contents = contents << 1;
            }
        }

        // sets the other flags
        ccr.x = ccr.c = last;
        contents = contents & 0 xffffffff ;
        ccr.v = 0;
        ccr.z = (contents == 0);
        ccr.n = (contents >> 31) & 0x0001;

        // This section stores the information back to the register again.
        d[reg].l = contents;
    }

    pc += 2;
}

void M68008::lsl_rMem() {

    // This method specifically only deals with word ops.

    short int dr = GETBITS( pc, 0x0100, 8 );
    short int reg = GETBITS( pc, 0x0007, 0 );
    short int mode = GETBITS( pc, 0x0038, 3 );
    short int contents = 0;

    // if <ea> is (an).
    if( mode == 2 ){
        // get the address from the address.
        contents = memory->GetWord( a[ reg ].w );

        // if it shifts right.
        if( dr == 0 ){
            ccr.x = ccr.c = contents & 0x0001;
            contents = contents >> 1;
        } else {
            // or shifting left.
```

```
        ccr.x = ccr.c = (contents & 0x8000) >> 15;
        contents = contents << 1;
    }

    // set the flags
    contents = contents & 0xffff;
    ccr.v = 0;
    ccr.z = (contents == 0);
    ccr.n = (contents >> 15) & 0x0001;

    // assigns the value back
    memory->SetWord( contents, a[ reg ].w );

    pc += 2;
}

// if <ea> is (an)+
if( mode == 3 ){

    // get the address from the address.
    contents = memory->GetWord( a[ reg ].w );

    // if it shifts right.
    if( dr == 0 ){
        ccr.x = ccr.c = contents & 0x0001;
        contents = contents >> 1;
    } else {
        // or shifting left.
        ccr.x = ccr.c = (contents & 0x8000) >> 15;
        contents = contents << 1;
    }

    // set the flags
    contents = contents & 0xffff;
    ccr.v = 0;
    ccr.z = (contents == 0);
    ccr.n = (contents >> 15) & 0x0001;

    // assigns the value back
    memory->SetWord( contents, a[ reg ].w );
    a[ reg ].w += 2;

    pc += 2;
}

// if <ea> is -(an)
if( mode == 4 ){

    // get the address from the address.
    contents = memory->GetWord( a[ reg ].w );

    // if it shifts right.
```

```
    if( dr == 0 ){
        ccr.x = ccr.c = contents & 0x0001;
        contents = contents >> 1;
    } else {
        // or shifting left.
        ccr.x = ccr.c = (contents & 0x8000) >> 15;
        contents = contents << 1;
    }

    // set the flags
    contents = contents & 0xffff;
    ccr.v = 0;
    ccr.z = (contents == 0);
    ccr.n = (contents >> 15) & 0x0001;

    // assigns the value back
    memory->SetWord( contents, a[ reg ].w );
    a[ reg ].w -= 2;

    pc += 2;
}

// if <ea> is d16(an)
if( mode == 5 ){
    int displ = (signed short int)memory->GetWord( pc + 2 )+ a[ reg ].w;
    // get the address from the address.
    contents = memory->GetWord( displ );

    // if it shifts right.
    if( dr == 0 ){
        ccr.x = ccr.c = contents & 0x0001;
        contents = contents >> 1;
    } else {
        // or shifting left.
        ccr.x = ccr.c = (contents & 0x8000) >> 15;
        contents = contents << 1;
    }

    // set the flags
    contents = contents & 0xffff;
    ccr.v = 0;
    ccr.z = (contents == 0);
    ccr.n = (contents >> 15) & 0x0001;

    // assigns the value back
    memory->SetWord( contents, displ );

    pc += 4;
}

// if <ea> is d8(an,Xn)
if( mode == 6 ){
```

```
    int displ = (signed char)memory->GetByte( pc + 3 )+ a[ reg ].w + d[ GETBITS( pc + 2, 0xf000, 12)
        ].l;
    // get the address from the address.
    contents = memory->GetWord( displ );

    // if it shifts right.
    if( dr == 0 ){
        ccr.x = ccr.c = contents & 0x0001;
        contents = contents >> 1;
    } else {
        // or shifting left.
        ccr.x = ccr.c = (contents & 0x8000) >> 15;
        contents = contents << 1;
    }

    // set the flags
    contents = contents & 0xffff;
    ccr.v = 0;
    ccr.z = (contents == 0);
    ccr.n = (contents >> 15) & 0x0001;

    // assigns the value back
    memory->SetWord( contents, displ );

    pc += 4;
}

// if <ea> is a real address.
if( mode == 7 ){
    long int address = 0;
    if( reg == 0 ){
        // word address
        address = memory->GetWord( pc + 2 );
        contents = memory->GetWord( address );

        // if it shifts right.
        if( dr == 0 ){
            ccr.x = ccr.c = contents & 0x0001;
            contents = contents >> 1;
        } else {
            // or shifting left.
            ccr.x = ccr.c = (contents & 0x8000) >> 15;
            contents = contents << 1;
        }

        // set the flags
        contents = contents & 0xffff;
        ccr.v = 0;
        ccr.z = (contents == 0);
        ccr.n = (contents >> 15) & 0x0001;

        // assigns the value back
```

```
            memory->SetWord( contents, address );

            pc += 4;

        }

        if( reg == 1 ){
            // long address
            address = memory->GetLongword( pc + 2 );
            contents = memory->GetWord( address );


            // if it shifts right.
            if( dr == 0 ){
                ccr.x = ccr.c = contents & 0x0001;
                contents = contents >> 1;
            } else {
                // or shifting left.
                ccr.x = ccr.c = (contents & 0x8000) >> 15;
                contents = contents << 1;
            }

            // set the flags
            contents = contents & 0xffff;
            ccr.v = 0;
            ccr.z = (contents == 0);
            ccr.n = (contents >> 15) & 0x0001;

            // assigns the value back
            memory->SetWord( contents, address );

            pc += 6;

        }
    }
}

void M68008::move()
{

    short int dreg, dmode, sreg, smode, size;
    unsigned long displ;
    char *buf;

    sreg = GETBITS( pc, 0x7, 0 );
    smode = GETBITS( pc, 0x38, 3 );
    dmode = GETBITS( pc, 0x1c0, 6 );
    dreg = GETBITS( pc, 0xe00, 9 );
    size = GETBITS( pc, 0x3000, 12 );

    DATA_REGISTER( information );
    char source[ 20 ]; // provides the source for the push operations.
```

```
// StackItem *stackItem;

    // if the source is a data register.
    if( smode == 0 ){
        sprintf( source, "d%X", sreg );

        // a byte operation
        if(size == 1) {
            information.b = d[ sreg ].b;
            SETNZ( information.b );
        }

        // a word operation
        if ( size == 3 ){
            information.w = d[ sreg ].w;
            SETNZ( information.w );
        }

        // a long operation
        if ( size == 2 ){
            information.l = d[ sreg ].l;
            SETNZ( information.l );
        }
        pc += 2;
    }

    // if the source is an address register
    if ( smode == 1 ){
        sprintf( source, "a%X", sreg );

        // a byte operation
        if ( size == 1 ){
            information.b = a[ sreg ].w & 0xff;
            SETNZ( information.b );
        }

        // a word operation
        if ( size == 3 ){
            information.w = a[ sreg ].w;
            SETNZ( information.w );
        }

        // a long operation
        if ( size == 2 ){
            information.l = a[ sreg ].l;
            SETNZ( information.l );
        }
        pc += 2;
    }

    // if the source is an (address pointer)
    if ( smode == 2 ){
```

```
        sprintf ( source, " (a%X)", sreg );

        // a byte operation
        if ( size == 1 ){
            information.b = memory->GetByte( a[ sreg ].l );
            SETNZ( information.b );
        }

        // a word operation
        if ( size == 3 ){
            information.w = memory->GetWord( a[ sreg ].l );
            SETNZ( information.w );
        }

        // a long operation
        if ( size == 2 ){
            information.l = memory->GetLongword( a[ sreg ].l );
            SETNZ( information.l );
        }
        pc += 2;
    }

    // if the source is an ( address pointer )+
    if ( smode == 3 ){
        sprintf ( source, " (a%X)+", sreg );

        // a byte operation
        if ( size == 1 ){
            information.b = memory->GetByte( a[ sreg ].l );
            SETNZ( information.b );
            a[ sreg ].l += 1;
        }

        // a word operation
        if ( size == 3 ){
            information.w = memory->GetWord( a[ sreg ].l );
            SETNZ( information.w );
            a[ sreg ].l += 2;
        }

        // a long operation
        if ( size == 2 ){
            information.l = memory->GetLongword( a[ sreg ].l );
            SETNZ( information.l );
            a[ sreg ].l += 4;
        }

        if ( sreg == 7 ){

            // if there are items on the stack.
            if ( systemStack.getCount() > 0 ){
```

```
            if ( (systemStack.getItem( 0 )->ID) !=
            (systemStack.getItem( size )->ID) ){

               // bad popping
               buf = new char[ 128 ];
               sprintf ( buf, "WARNING at address $%lX: The size of the pop is not the same\n as the size
                    that was pushed on. Check your Stack use.", pc );
               helpStack.push( buf );

            }

            if ( size == 1 ) {
               POP(1);
            }

            if ( size == 3) {
               POP(2);
            }

            if ( size == 2 ) {
               POP(4);
            }

         }
         else
         {

            // Stack underflow operation is occuring.
            buf = new char[ 128 ];
            sprintf ( buf, "STACK UNDERFLOW at address $%lX: You are trying to pop a value\n off the
                 stack that is not there. This will cause a bus error.", pc );
            helpStack.push( buf );
         }
      }

      pc += 2;
   }

   // if the source is an -( address pointer )
   if ( smode == 4 ){
      sprintf ( source, "-(a%X)", sreg );


      // a byte operation
      if ( size == 1 ) {
         a[ sreg ].l -= 1;
         information.b = memory->GetByte( a[ sreg ].l );
         SETNZ( information.b );
      }
```

```
    // a word operation
    if ( size == 3 ){
       a[ sreg ].l -= 2;
       information.w = memory->GetWord( a[ sreg ].l );
       SETNZ( information.w );
       }

    // a long operation
    if ( size == 2 ){
       a[ sreg ].l -= 4;
       information.l = memory->GetLongword( a[ sreg ].l );
       SETNZ( information.l );
    }
    pc += 2;
}


// d16(An)
// if the source is a displacement to address register
if ( smode == 5 ){
    sprintf( source, "$%X(a%X)", (signed short int)memory->GetWord( pc + 2 ), sreg );

    displ = (signed short int)memory->GetWord( pc + 2 )+ a[ sreg ].l;

    // a byte operation
    if ( size == 1 ){

       information.b = memory->GetByte( displ );
       SETNZ( information.b );
    }

    // a word operation
    if ( size == 3 ){
       information.w = memory->GetWord( displ );
       SETNZ( information.w );
       }

    // a long operation
    if ( size == 2 ){
       information.l = memory->GetLongword( displ );
       SETNZ( information.l );
    }
    pc += 4;
}

// d8(An, Xn)
// if the source is a displacement to address register , and data register
if ( smode == 6 ){

    sprintf( source, "$%X(a%X,d%X)", (signed short int)memory->GetByte( pc + 3 ), sreg, GETBITS(
        pc + 2, 0xF000, 12 ));
```

```
    displ = (signed char)memory->GetByte( pc + 3 )+ a[ sreg ].l
        + d[ GETBITS( pc + 2, 0xF000, 12 )].l;


    // a byte operation
    if ( size == 1 ){
        information.b = memory->GetByte( displ );
        SETNZ( information.b );
    }

    // a word operation
    if ( size == 3 ){
        information.w = memory->GetWord( displ );
        SETNZ( information.w );
        }

    // a long operation
    if ( size == 2 ){
        information.l = memory->GetLongword( displ );
        SETNZ( information.l );
    }
    pc += 4;
}

// mode is 7
if ( smode == 7 ){

    // (XXX).W
    if ( sreg == 0 ){

        displ = memory->GetWord( pc + 2 );

        sprintf( source, "$%lX", displ );

        // a byte operation
        if ( size == 1 ){
            information.b = memory->GetByte( displ );
            SETNZ( information.b );
        }

        // a word operation
        if ( size == 3 ){
            information.w = memory->GetWord( displ );
            SETNZ( information.w );
            }

        // a long operation
        if ( size == 2 ){
            information.l = memory->GetLongword( displ );
            SETNZ( information.l );
        }
        pc += 4;
```

```
}

// (XXX),L
if ( sreg == 1 ){

    displ = memory->GetLongword( pc + 2 );

    sprintf ( source, "$%lX", displ );

    // a byte operation
    if ( size == 1 ){
        information.b = memory->GetByte( displ );
        SETNZ( information.b );
    }

    // a word operation
    if ( size == 3 ){
        information.w = memory->GetWord( displ );
        SETNZ( information.w );
        }

    // a long operation
    if ( size == 2 ){
        information.l = memory->GetLongword( displ );
        SETNZ( information.l );
    }
    pc += 6;
}

if ( sreg == 4 ){
    sprintf ( source, "#immediate" );

    if ( size == 1 ){
        // a byte operation
        information.w = memory->GetWord( pc + 2 );
        SETNZ( information.b );
        pc += 4;
    }

    if ( size == 3 ){
        // a word operation
        information.w = memory->GetWord( pc + 2 );
        SETNZW( information.w );
        pc += 4;
    }

    if ( size == 2 ){
        // a longword operation
        information.l = memory->GetLongword( pc + 2 );
        SETNZL( information.l );
        pc += 6;
    }
```

```
    }
}


/* OMIT--------
   d16(PC) and d8(PC, Xn)
   */



//-------------------------------------------------------------------//


// destination is a data register
if( dmode == 0 ){

   // a byte operation
   if(size == 1) {
      d[ dreg ].b = information.b;
   }

   // a word operation
   if ( size == 3 ){
      d[ dreg ].w = information.w;
   }

   // a long operation
   if ( size == 2 ){
      d[ dreg ].l = information.l;
   }
}

// destination is an address register pointer
if ( dmode == 2 ){

   // a byte operation
   if(size == 1) {
      memory->SetByte( information.b, a[ dreg ].l );
   }

   // a word operation
   if ( size == 3 ){
      memory->SetWord( information.w, a[ dreg ].l );
   }

   // a long operation
   if ( size == 2 ){
      memory->SetLongword( information.l, a[ dreg ].l );
   }
}

// destination is an (address pointer)+
```

```
if ( dmode == 3 ){

    // a byte operation
    if(size == 1) {
        memory->SetByte( information.b, a[ dreg ].l );
        a[ dreg ].l += 1;
    }

    // a word operation
    if ( size == 3 ){
        memory->SetWord( information.w, a[ dreg ].l );
        a[ dreg ].l += 2;
    }

    // a long operation
    if ( size == 2 ){
        memory->SetLongword( information.l, a[ dreg ].l );
        a[ dreg ].l += 4;
    }
}

// destination is an -(address pointer)
if ( dmode == 4 ){

    // a byte operation
    if(size == 1) {
        a[ dreg ].l -= 1;
        memory->SetByte( information.b, a[ dreg ].l );
    }

    // a word operation
    if ( size == 3 ){
        a[ dreg ].l -= 2;
        memory->SetWord( information.w, a[ dreg ].l );
    }

    // a long operation
    if ( size == 2 ){
        a[ dreg ].l -= 4;
        memory->SetLongword( information.l, a[ dreg ].l );
    }

    if( dreg == 7 ){
        if(size == 1) {
            PUSH(source,1);
        }

        // a word operation
        if ( size == 3 ){
            PUSH(source,2);
        }
```

```
        // a long operation
        if ( size == 2 ){
            PUSH(source,4);
        }
    }

    // ! If the register is a7 then this is a stack operation
    // and should be displayed on the stack viewer as a push op.

}

// destination is displacement to address pointer
if ( dmode == 5 ){

    displ = (signed short int)memory->GetWord( pc )+ a[ dreg ].l;

    // a byte operation
    if(size == 1) {
        memory->SetByte( information.b, displ );

    }

    // a word operation
    if ( size == 3 ){
        memory->SetWord( information.w, displ );
    }

    // a long operation
    if ( size == 2 ){
        memory->SetLongword( information.l, displ );
    }
    pc += 2;
}

// destination is displacement to address pointer and data registers
if ( dmode == 6 ){

    displ = (signed char)memory->GetByte( pc + 1 )+ a[ sreg ].l
        + d[ GETBITS( pc, 0xF000, 12 )].l;


    // a byte operation
    if(size == 1) {
        memory->SetByte( information.b, displ );

    }

    // a word operation
    if ( size == 3 ){
        memory->SetWord( information.w, displ );
    }
```

```
    // a long operation
    if ( size == 2 ){
       memory->SetLongword( information.l, displ );
    }
    pc += 2;
}


if ( dmode == 7 ){

    // destination is (XXX).W
    if ( dreg == 0 ){

       displ = memory->GetWord( pc );

       // a byte operation
       if(size == 1) {
          memory->SetByte( information.b, displ );

       }

       // a word operation
       if ( size == 3 ){
          memory->SetWord( information.w, displ );
       }

       // a long operation
       if ( size == 2 ){
          memory->SetLongword( information.l, displ );
       }
       pc += 2;
    }

    // destination is (XXX).L
    if ( dreg == 1 ){

       displ = memory->GetLongword( pc );

       // a byte operation
       if(size == 1) {
          memory->SetByte( information.b, displ );

       }

       // a word operation
       if ( size == 3 ){
          memory->SetWord( information.w, displ );
       }

       // a long operation
       if ( size == 2 ){
          memory->SetLongword( information.l, displ );
```

```
            }
            pc += 4;
        }
    }
    // V and C are cleared.
    ccr.v = ccr.c = 0;
}

void M68008::movea()
{

    short int dreg, sreg, smode, size;
    unsigned long displ;
    unsigned long int oldPC = pc;
    char *buf;

    sreg = GETBITS( pc, 0x7, 0 );
    smode = GETBITS( pc, 0x38, 3 );
    dreg = GETBITS( pc, 0xe00, 9 );
    size = GETBITS( pc, 0x3000, 12 );

    signed long information;

    //source
    if( smode == 0 ){

        // a word operation
        if ( size == 3 ){
            information = ( signed short int )d[ sreg ].w;
            SETNZ( information );
        }

        // a long operation
        if ( size == 2 ){
            information = ( signed long )d[ sreg ].l;
            SETNZ( information );
        }
        pc += 2;
    }

    // if the source is an address register
    if ( smode == 1 ){

        // a word operation
        if ( size == 3 ){
            information = ( signed short int )a[ sreg ].w;
            SETNZ( information );
        }

        // a long operation
        if ( size == 2 ){
            information = ( signed long )a[ sreg ].l;
```

```
        SETNZ( information );
    }
    pc += 2;
}

// if the source is an address pointer
if ( smode == 2 ){

    // a word operation
    if ( size == 3 ){
        information = ( signed short int )memory->GetWord( a[ sreg ].l );
        SETNZ( information );
    }

    // a long operation
    if ( size == 2 ){
        information = ( signed long )memory->GetLongword( a[ sreg ].l );
        SETNZ( information );
    }
    pc += 2;
}

// if the source is an ( address pointer )+
if ( smode == 3 ){

    // a word operation
    if ( size == 3 ){
        information = ( signed short int )memory->GetWord( a[ sreg ].l );
        SETNZ( information );
        a[ sreg ].l += 2;
        POP( 2 );
    }

    // a long operation
    if ( size == 2 ){
        information = ( signed long )memory->GetLongword( a[ sreg ].l );
        SETNZ( information );
        a[ sreg ].l += 4;
        POP( 4 );
    }
    pc += 2;
}

// if the source is an -( address pointer )
if ( smode == 4 ){

    // a word operation
    if ( size == 3 ){
        a[ sreg ].l -= 2;
        information = ( signed short int )memory->GetWord( a[ sreg ].l );
        SETNZ( information );
        }
```

```
    // a long operation
    if ( size == 2 ){
        a[ sreg ].l -= 4;
        information = ( signed long )memory->GetLongword( a[ sreg ].l );
        SETNZ( information );
    }
    pc += 2;
}


// d16(An)
// if the source is a displacement to address register
if ( smode == 5 ){

    displ = (signed short int)memory->GetWord( pc + 2 )+ a[ sreg ].l;

    // a word operation
    if ( size == 3 ){
        information = ( signed short int )memory->GetWord( displ );
        SETNZ( information );
        }

    // a long operation
    if ( size == 2 ){
        information = ( signed long )memory->GetLongword( displ );
        SETNZ( information );
    }
    pc += 4;
}

// d8(An, Xn)
// if the source is a displacement to address register , and data register
if ( smode == 6 ){

    displ = (signed char)memory->GetByte( pc + 3 )+ a[ sreg ].l
        + d[ GETBITS( pc + 2, 0xF000, 12 )].l;

    // a word operation
    if ( size == 3 ){
        information = ( signed short int )memory->GetWord( displ );
        SETNZ( information );
        }

    // a long operation
    if ( size == 2 ){
        information = ( signed long )memory->GetLongword( displ );
        SETNZ( information );
    }
    pc += 4;
}
```

```
// mode is 7
if ( smode == 7 ){

   // (XXX).W
   if ( sreg == 0 ){

      displ = memory->GetWord( pc + 2 );

      // a word operation
      if ( size == 3 ){
         information = ( signed short int )memory->GetWord( displ );
         SETNZ( information );
         }

      // a long operation
      if ( size == 2 ){
         information = ( signed long )memory->GetLongword( displ );
         SETNZ( information );
      }
      pc += 4;
   }

   // (XXX),L
   if ( sreg == 1 ){

      displ = memory->GetLongword( pc + 2 );

      // a word operation
      if ( size == 3 ){
         information = ( signed short int )memory->GetWord( displ );
         SETNZ( information );
         }

      // a long operation
      if ( size == 2 ){
         information = ( signed long )memory->GetLongword( displ );
         SETNZ( information );
      }
      pc += 6;
   }

   if ( sreg == 4 ){
      if ( size == 3 )
      {
         // a word operation
         information = memory->GetWord( pc + 2 );
         SETNZW( information );
         pc += 4;
      }

      if ( size == 2 )
      {
```

```
            // a longword operation
            information = memory->GetLongword( pc + 2 );
            SETNZL( information );
            pc += 6;
         }
      }
   }


   /* OMIT-------
      d16(PC) and d8(PC, Xn)
   */

   // destination
   // destination is an address register
   // a word operation
   if ( size == 3 ){
      a[ dreg ].l = information;

      if( smode == 7 )
      {
         // source is a word or lword from memory, destionation is an address register
         buf = new char[ 128 ];
         sprintf( buf, "WARNING at address $%lX: You're moving a word or longword from memory into an "
            "address register.\nYou may have wanted to move an immediate value instead (forgot "
            "the hash?", oldPC );
         helpStack.push( buf );
      }
   }

   // a long operation
   if ( size == 2 ){
      a[ dreg ].l = information;
   }

   if( dreg == 7 )
   {
      buf = new char[ 128 ];
      sprintf( buf, "WARNING at address $%lX: You probably shouldn't\n edit the SSP (a7) directly.",
         oldPC );
      helpStack.push( buf );
   }
}

void M68008::moveFromCcr()
{
   short int reg, mode;
   unsigned long displ;
   reg = GETBITS( pc, 0x7, 0 );
   mode = GETBITS( pc, 0x38, 3 );
   DATA_REGISTER( result );
```

```
// only one size ... word
// must be zero-extended to a word size
result.w = ccr.c + ( ccr.v << 1 )+ ( ccr.z << 2 )+ ( ccr.n << 3 )+ ( ccr.x << 4 );


// -> dn
if ( mode == 0 ){
    d[ reg ].w = result.w;
    pc += 2;
}

// -> (An)
if ( mode == 2 ){
    long address = a[ reg ].l;
    memory->SetWord( result.w, memory->GetLongword( address ));
    pc += 2;
}

// -> (An)+
if ( mode == 3 ){
    long address = a[ reg ].l;
    memory->SetWord( result.w, memory->GetLongword( address ));
    a[ reg ].l += 2;
    pc += 2;
}

// -> -(An)
if ( mode == 4 ){
    a[ reg ].l -= 2;
    long address = a[ reg ].l;
    memory->SetWord( result.w, memory->GetLongword( address ));
    pc += 2;
}

// -> (d16, An)
if ( mode == 5 ){
    displ = (signed short int)memory->GetWord( pc + 2 )+ a[ reg ].l;

    memory->SetWord( result.w, displ );
    pc += 4;
}

// -> d8(An, Xn)
if ( mode == 6 ){
    displ = (signed char)memory->GetByte( pc + 3 )+ a[ reg ].l
        + d[ GETBITS( pc + 2, 0xF000, 12 )].l;

    memory->SetWord( result.w, displ );
    pc += 4;
}
```

```
    // -> (XXX).W
    if ( mode == 7 && reg == 0 ){
        displ = memory->GetWord( pc + 2 );
        memory->SetWord( result.w, displ );
        pc += 4;
    }

    // -> (XXX).L
    if ( mode == 7 && reg == 1 ){
        displ = memory->GetLongword( pc + 2 );
        memory->SetWord( result.w, displ );
        pc += 6;
    }
}

void M68008::moveToCcr()
{
    // always a word operation

    short int operand;
    short int reg, mode;
    unsigned long displ;
    reg = GETBITS( pc, 0x7, 0 );
    mode = GETBITS( pc, 0x38, 3 );

    // (Dn) -> CCR
    if ( mode == 0 ){
        operand = d[ reg ].w & 0x1f;
        pc += 2;
    }


    // (An) -> CCR
    if ( mode == 2 ){
        long address = a[ reg ].l;
        operand = memory->GetWord( address );
        pc += 2;
    }

    // (An)+ -> CCR
    if ( mode == 3 ){
        long address = a[ reg ].l;
        operand = memory->GetWord( address );
        a[ reg ].l += 2;
        pc += 2;
    }

    // -(An) -> CCR
    if ( mode == 4 ){
        a[ reg ].l -= 2;
        long address = a[ reg ].l;
        operand = memory->GetWord( address );
```

```
        pc += 2;
    }

    // (d16, An) -> CCR
    if ( mode == 5 ){
        displ = (signed short int)memory->GetWord( pc + 2 )+ a[ reg ].l;
        operand = memory->GetWord( displ );
        pc += 4;
    }

    // d8(An, Xn) -> CCR
    if ( mode == 6 ){
        displ = (signed char)memory->GetByte( pc + 3 )+ a[ reg ].l
            + d[ GETBITS( pc + 2, 0xF000, 12 )].l;
        operand = memory->GetWord( displ );
        pc += 4;
    }

    // (XXX).W -> CCR
    if ( mode == 7 && reg == 0 ){
        displ = memory->GetWord( pc + 2 );
        operand = memory->GetWord( displ );
        pc += 4;
    }

    // (XXX).L -> CCR
    if ( mode == 7 && reg == 1 ){
        displ = memory->GetLongword( pc + 2 );
        operand = memory->GetWord( displ );
        pc += 6;
    }

    // #<data> -> CCR
    if ( mode == 7 && reg == 4 ){
        operand = memory->GetWord( pc + 2 );
        pc += 4;
    }

    ccr.x = ( operand & 0x10 )>> 4 ;
    ccr.n = ( operand & 0x8 )>> 3;
    ccr.z = ( operand & 0x4 )>> 2;
    ccr.v = ( operand & 0x2 )>> 1;
    ccr.c = operand & 0x1;

    /* OMIT: (d16, PC), d8(PC, Xn) */
}

void M68008::movem()
{
short int dr, size , mode, reg;
    dr = GETBITS( pc, 0x400, 10 );
    size = GETBITS( pc, 0x40, 6 );
```

```
   mode = GETBITS( pc, 0x38, 3 );
   reg = GETBITS( pc, 0x7, 0 );
   short int mask = memory->GetWord( pc + 2 );
   short int count = 0;
   long int address;


   // to calculate the number of registers marked
   for ( int i = 0; i < 16; i++ ){
      int temp = mask >> i;
      temp = temp & 0x0001;

      if( temp == 1 ){
         count++;
      }
   }

   /* SOURCE: check the direction of the transfer: 1 if memory to register
   so if dr is 1 then take the required data from memory
   */
   if ( dr == 1 )
   {
      //
      // This is the section where we extract the address.

      // check addressing mode
      // (an)
      if ( mode == 2 )
      {
         address = a[ reg ].l;
      }

      // (an)+
      if ( mode == 3 )
      {
         address = a[ reg ].l;

/*       // Changes address to point to the start of the information.
         if( size == 0 ){
            // if it is a word
            address += count * 2;
         } else {
            // if it is a long word
            address += count * 4;
         }
*/

      }

      // d16(An)
      if ( mode == 4 )
      {
```

```
    address = ( signed short int )memory->GetWord( pc + 4 )+ a[ reg ].l;
}

// d8(An, dn)
if ( mode == 5 )
{
    address = ( signed short int )GETBITS( pc + 4, 0x0fff, 0 )+ a[ reg ].l + d[ GETBITS( pc + 4, 0
        xf000, 12 )].l;
}

if ( mode == 7 )
{
    // (XXX).w
    if ( reg == 0 )
    {
        address = memory->GetWord( pc + 4 );
    }

    // (XXX).L
    if ( reg == 1 )
    {
        address = memory->GetLongword( pc + 4 );
    }
}

// REGISTERS: we then save the data into the set registers
// each bit will have to be checked to see if it is true.

if ( ( mask & 0x0001 )!= 0 ) {
    // d0
    if( size == 0 ){
        // word operation
        signed short int contents = d[ 0 ].w;
        memory->SetLongword( contents, address );
        address -= 4;
    } else {
        // long operation
        memory->SetLongword( d[0].l, address );
        address -= 4;
    }

}

if ( ( mask & 0x0002 )!= 0 ) {
    // d1
    if( size == 0 ){
        // word operation
        signed short int contents = d[ 1 ].w;
        memory->SetLongword( contents, address );
        address -= 4;
    } else {
        // long operation
```

```
        memory->SetLongword( d[ 1 ].l, address );
        address -= 4;
    }
}

if ( ( mask & 0x0004 )!= 0 ) {
    // d2
    if( size == 0 ){
        // word operation
        signed short int contents = d[ 2 ].w;
        memory->SetLongword( contents, address );
        address -= 4;
    } else {
        // long operation
        memory->SetLongword( d[ 2 ].l, address );
        address -= 4;
    }
}

if ( ( mask & 0x0008 )!= 0 ) {
    // d3
    if( size == 0 ){
        // word operation
        signed short int contents = d[ 3 ].w;
        memory->SetLongword( contents, address );
        address -= 4;
    } else {
        // long operation
        memory->SetLongword( d[ 3 ].l, address );
        address -= 4;
    }
}

if ( ( mask & 0x0010 )!= 0 ) {
    // d4
    if( size == 0 ){
        // word operation
        signed short int contents = d[ 4 ].w;
        memory->SetLongword( contents, address );
        address -= 4;
    } else {
        // long operation
        memory->SetLongword( d[ 4 ].l, address );
        address -= 4;
    }
}

if ( ( mask & 0x0020 )!= 0 ) {
    // d5
    if( size == 0 ){
        // word operation
        signed short int contents = d[ 5 ].w;
```

```
            memory->SetLongword( contents, address );
            address -= 4;
        } else {
            // long operation
            memory->SetLongword( d[ 5 ].l, address );
            address -= 4;
        }
    }

    if ( ( mask & 0x0040 )!= 0 ) {
        // d6
        if ( size == 0 ) {
            // word operation
            signed short int contents = d[ 6 ].w;
            memory->SetLongword( contents, address );
            address -= 4;
        } else {
            // long operation
            memory->SetLongword( d[ 6 ].l, address );
            address -= 4;
        }
    }

    if ( ( mask & 0x0080 )!= 0 ) {
        // d7
        if ( size == 0 ) {
            // word operation
            signed short int contents = d[ 7 ].w;
            memory->SetLongword( contents, address );
            address -= 4;
        } else {
            // long operation
            memory->SetLongword( d[ 7 ].l, address );
            address -= 4;
        }
    }

    if ( ( mask & 0x0100 )!= 0 ) {
        // a0
        if ( size == 0 ) {
            // word operation
            signed short int contents = a[ 0 ].w;
            memory->SetLongword( contents, address );
            address -= 4;
        } else {
            // long operation
            memory->SetLongword( a[ 0 ].l, address );
            address -= 4;
        }
    }

    if ( ( mask & 0x0200 )!= 0 ) {
```

```
    // a1
    if( size == 0 ){
        // word operation
        signed short int contents = a[ 1 ].w;
        memory->SetLongword( contents, address );
        address -= 4;
    } else {
        // long operation
        memory->SetLongword( a[ 1 ].l, address );
        address -= 4;
    }
}

if ( ( mask & 0x0400 )!= 0 ) {
    // a2
    if( size == 0 ){
        // word operation
        signed short int contents = a[ 2 ].w;
        memory->SetLongword( contents, address );
        address -= 4;
    } else {
        // long operation
        memory->SetLongword( a[ 2 ].l, address );
        address -= 4;
    }
}

if ( ( mask & 0x0800 )!= 0 ) {
    // a3
    if( size == 0 ){
        // word operation
        signed short int contents = a[ 3 ].w;
        memory->SetLongword( contents, address );
        address -= 4;
    } else {
        // long operation
        memory->SetLongword( a[ 3 ].l, address );
        address -= 4;
    }
}

if ( ( mask & 0x1000 )!= 0 ) {
    // a4
    if( size == 0 ){
        // word operation
        signed short int contents = a[ 4 ].w;
        memory->SetLongword( contents, address );
        address -= 4;
    } else {
        // long operation
        memory->SetLongword( a[ 4 ].l, address );
        address -= 4;
```

```
        }
    }

    if ( ( mask & 0x2000 )!= 0 ) {
        // a5
        if ( size == 0 ) {
            // word operation
            signed short int contents = a[ 5 ].w;
            memory->SetLongword( contents, address );
            address -= 4;
        } else {
            // long operation
            memory->SetLongword( a[ 5 ].l, address );
            address -= 4;
        }
    }

    if ( ( mask & 0x4000 )!= 0 ) {
        // a6
        if ( size == 0 ) {
            // word operation
            signed short int contents = a[ 6 ].w;
            memory->SetLongword( contents, address );
            address -= 4;
        } else {
            // long operation
            memory->SetLongword( a[ 6 ].l, address );
            address -= 4;
        }
    }

    if ( ( mask & 0x8000 )!= 0 ) {
        // a7
        if ( size == 0 ) {
            // word operation
            signed short int contents = a[ 7 ].w;
            memory->SetLongword( contents, address );
            address -= 4;
        } else {
            // long operation
            memory->SetLongword( a[ 7 ].l, address );
            address -= 4;
        }
    }

    if ( (mode == 3) && ( reg == 7 )){
        a[ reg ]. l = address;
    }

} else {

    //
```

```
// Section so extract the address where the information is at.

      // check addressing mode
// (an)
if ( mode == 2 )
{
   address = a[ reg ].l;
}

// (an)+
if ( mode == 3 )
{
   address = a[ reg ].l;

   // REGISTERS: we then save the data into the set registers
   // each bit will have to be checked to see if it is true.
   if ( ( mask & 0x0001 )!= 0 ) {
      // a7
      a [ 7 ].l = memory->GetLongword( address );
      address += 4;
   }

   if ( ( mask & 0x0002 )!= 0 ) {
      // a6
      a [ 6 ].l = memory->GetLongword( address );
      address += 4;
   }

   if ( ( mask & 0x0004 )!= 0 ) {
      // a5
      a [ 5 ].l = memory->GetLongword( address );
      address += 4;
   }

   if ( ( mask & 0x0008 )!= 0 ) {
      // a4
      a [ 4 ].l = memory->GetLongword( address );
      address += 4;
   }

   if ( ( mask & 0x0010 )!= 0 ) {
      // a3
      a [ 3 ].l = memory->GetLongword( address );
      address += 4;
   }

   if ( ( mask & 0x0020 )!= 0 ) {
      // a2
      a [ 2 ].l = memory->GetLongword( address );
      address += 4;
   }
```

```
if ( ( mask & 0x0040 )!= 0 ) {
   // a1
   a [ 1 ]. l = memory->GetLongword( address );
   address += 4;
}

if ( ( mask & 0x0080 )!= 0 ) {
   // a0
   a [ 0 ]. l = memory->GetLongword( address );
   address += 4;
}

if ( ( mask & 0x0100 )!= 0 ) {
   // d7
   d [ 7 ]. l = memory->GetLongword( address );
   address += 4;
}

if ( ( mask & 0x0200 )!= 0 ) {
   // d6
   d [ 6 ]. l = memory->GetLongword( address );
   address += 4;
}

if ( ( mask & 0x0400 )!= 0 ) {
   // d5
   d [ 5 ]. l = memory->GetLongword( address );
   address += 4;
}

if ( ( mask & 0x0800 )!= 0 ) {
   // d4
   d [ 4 ]. l = memory->GetLongword( address );
   address += 4;
}

if ( ( mask & 0x1000 )!= 0 ) {
   // d3
   d [ 3 ]. l = memory->GetLongword( address );
   address += 4;
}

if ( ( mask & 0x2000 )!= 0 ) {
   // d2
   d [ 2 ]. l = memory->GetLongword( address );
   address += 4;
}

if ( ( mask & 0x4000 )!= 0 ) {
   // d1
   d [ 1 ]. l = memory->GetLongword( address );
   address += 4;
```

```
    }

    if ( ( mask & 0x8000 )!= 0 ) {
        // d0
        d [ 0 ]. l = memory->GetLongword( address );
        address += 4;
    }

    if( reg == 7 ){
        a [ 7 ]. l = address;
    }


    return;
}

// d16(An)
if ( mode == 4 )
{
    address = ( signed short int )memory->GetWord( pc + 4 )+ a[ reg ].l;
}

// d8(An, dn)
if ( mode == 5 )
{
    address = ( signed short int )GETBITS( pc + 4, 0x0fff, 0 )+ a[ reg ].l + d[ GETBITS( pc + 4, 0
        xf000, 12 )].l;
}

if ( mode == 7 )
{
    // (XXX).w
    if ( reg == 0 )
    {
        address = memory->GetWord( pc + 4 );
    }

    // (XXX).L
    if ( reg == 1 )
    {
        address = memory->GetLongword( pc + 4 );
    }
}



// REGISTERS: we then save the data into the set registers
// each bit will have to be checked to see if it is true.
if ( ( mask & 0x0001 )!= 0 ) {
    // d0
    d [ 0 ]. l = memory->GetLongword( address );
    address += 4;
```

```
}

if ( ( mask & 0x0002 )!= 0 ) {
   // d1
   d [ 1 ]. l = memory->GetLongword( address );
   address += 4;
}

if ( ( mask & 0x0004 )!= 0 ) {
   // d2
   d [ 2 ]. l = memory->GetLongword( address );
   address += 4;
}

if ( ( mask & 0x0008 )!= 0 ) {
   // d3
   d [ 3 ]. l = memory->GetLongword( address );
   address += 4;
}

if ( ( mask & 0x0010 )!= 0 ) {
   // d4
   d [ 4 ]. l = memory->GetLongword( address );
   address += 4;
}

if ( ( mask & 0x0020 )!= 0 ) {
   // d5
   d [ 5 ]. l = memory->GetLongword( address );
   address += 4;
}

if ( ( mask & 0x0040 )!= 0 ) {
   // d6
   d [ 6 ]. l = memory->GetLongword( address );
   address += 4;
}

if ( ( mask & 0x0080 )!= 0 ) {
   // d7
   d [ 7 ]. l = memory->GetLongword( address );
   address += 4;
}

if ( ( mask & 0x0100 )!= 0 ) {
   // a0
   a [ 0 ]. l = memory->GetLongword( address );
   address += 4;
}

if ( ( mask & 0x0200 )!= 0 ) {
   // a1
```

```
            a [ 1 ]. l = memory->GetLongword( address );
            address += 4;
        }

        if ( ( mask & 0x0400 )!= 0 ) {
            // a2
            a [ 2 ]. l = memory->GetLongword( address );
            address += 4;
        }

        if ( ( mask & 0x0800 )!= 0 ) {
            // a3
            a [ 3 ]. l = memory->GetLongword( address );
            address += 4;
        }

        if ( ( mask & 0x1000 )!= 0 ) {
            // a4
            a [ 4 ]. l = memory->GetLongword( address );
            address += 4;
        }

        if ( ( mask & 0x2000 )!= 0 ) {
            // a5
            a [ 5 ]. l = memory->GetLongword( address );
            address += 4;
        }

        if ( ( mask & 0x4000 )!= 0 ) {
            // a6
            a [ 6 ]. l = memory->GetLongword( address );
            address += 4;
        }

        if ( ( mask & 0x8000 )!= 0 ) {
            // a7
            a [ 7 ]. l = memory->GetLongword( address );
            address += 4;
        }
    }

    /* DESTINATION: check the direction. if 1 then no need to do anything.
    If 0 then must store data from registers into memory.
    */

}

void M68008::movep()
{
}

void M68008::moveq()
```

```
{
    // size is always longword
    char data = GETBITS( pc, 0xFF, 0 );
    short int reg = GETBITS( pc, 0xE00, 9 );

    d[ reg ].l = ( signed char )data;
    SETNZ( d[ reg ].l );
    pc += 2;


}

void M68008::mulsWord()
{

    unsigned long displ;
    short int sreg, smode, dreg;
    sreg = GETBITS( pc, 0x7, 0 );
    smode = GETBITS( pc, 0x38, 3 );
    // destination is always a register
    dreg = GETBITS( pc, 0xE00, 9 );

    // carry bit is always cleared
    ccr.c = 0;
    // overflow only occurs when multiplying 32 bit operands
    ccr.v = 0;

    // dn x dn -> dn
    if ( smode == 0 ){
        d[ dreg ].l = ( signed short int )d[ dreg ].w * ( signed short int )d[ sreg ].w;
        pc += 2;
        SETNZL( d[ dreg ].l );
    }

    // dn x (An) -> dn
    if ( smode == 2 ){
        long address = a[ sreg ].l;
        d[ dreg ].l = ( ( signed short int )d[ dreg ].w
            * ( signed short int )memory->GetWord( address ));
        pc += 2;
        SETNZ( d[ dreg ].l );
    }

    // dn x (An)+ -> dn
    if ( smode == 3 ){
        long address = a[ sreg ].l;
        d[ dreg ].l = ( ( signed short int )d[ dreg ].w
            * ( signed short int )memory->GetWord( address ));
        a[ sreg ].l += 2;
        pc += 2;
        SETNZ( d[ dreg ].l );
    }
```

```
// dn x -(An) -> dn
if (smode == 4 ){
    a[ sreg ].l -= 2;
    long address = a[ sreg ].l;
    d[ dreg ].l = ( ( signed short int )d[ dreg ].w
        * ( signed short int )memory->GetWord( address ));
    pc += 2;
    SETNZ( d[ dreg ].l );
}

// dn x d16(An) -> dn
if (smode == 5 ){
    displ = (signed short int)memory->GetWord( pc + 2 )+ a[ sreg ].l;
    d[ dreg ].l = ( signed short int )d[ dreg ].w
        * ( signed short int )memory->GetWord( displ );
    SETNZ( d[ dreg ].l );
    pc += 4;
}

// dn x d8(An, Xn) -> dn
if ( smode == 6 ){
    displ = (signed char)memory->GetByte( pc + 3 )+ a[ sreg ].l
        + d[ GETBITS( pc + 2, 0xF000, 12 )].l;
    d[ dreg ].l = ( signed short int )d[ dreg ].w
        * ( signed short int )memory->GetWord( displ );
    SETNZ( d[ dreg ].l );
    pc += 4;
}

// dn x (xxx).w -> dn
if ( smode == 7 && sreg == 0 ){
    displ = memory->GetWord( pc + 2 );
    d[ dreg ].l = ( signed short int )d[ dreg ].w
        * ( signed short int )memory->GetWord( displ );
    SETNZ( d[ dreg ].l );
    pc += 4;
}

// dn x (xxx).L -> dn
if (smode == 7 && sreg == 1 ){
    displ = memory->GetLongword( pc + 2 );
    d[ dreg ].l = ( signed short int )d[ dreg ]. w
        * ( signed short int )memory->GetWord( displ );
    SETNZ( d[ dreg ].l );
    pc += 6;
}

// dn x #<xxx> -> dn
if ( smode == 7 && sreg == 4 ){
    displ = memory->GetWord( pc + 2 );
    d[ dreg ].l = ( signed short int )d[ dreg ].w
```

```
            * ( signed short int )displ;
        SETNZ( d[ dreg ].l );
        pc += 4;
    }


    /*
    OMIT: d16(PC) and d8(An, Xn)
    */
}

void M68008::mulsLong()
{
    // Invalid Size
}

void M68008::muluWord()
{
    unsigned long displ;
    short int sreg, smode, dreg;
    sreg = GETBITS( pc, 0x7, 0 );
    smode = GETBITS( pc, 0x38, 3 );
    // destination is always a register
    dreg = GETBITS( pc, 0xE00, 9 );

    // carry bit is always cleared
    ccr.c = 0;
    // overflow only occurs when multiplying 32 bit operands
    ccr.v = 0;

    // dn x dn -> dn
    if ( smode == 0 ){
        d[ dreg ].l = d[ dreg ].w * ( unsigned short int )d[ sreg ].w;
        pc += 2;
        SETNZL( d[ dreg ].l );
    }

    // dn x (An) -> dn
    if ( smode == 2 ){
        long address = a[ sreg ].l;
        d[ dreg ].l = d[ dreg ].w * ( unsigned short int )memory->GetWord( address );
        pc += 2;
        SETNZL( d[ dreg ].l );
    }

    // dn x (An)+ -> dn
    if ( smode == 3 ){
        long address = a[ sreg ].l;
        d[ dreg ].l = d[ dreg ].w * ( unsigned short int )memory->GetWord( address );
        a[ sreg ].l += 2;
        pc += 2;
        SETNZL( d[ dreg ].l );
```

```
}

// dn x -(An) -> dn
if (smode == 4 ){
    a[ sreg ].l -= 2;
    long address = a[ sreg ].l;
    d[ dreg ].l = d[ dreg ].w * ( unsigned short int )memory->GetWord( address );
    pc += 2;
    SETNZL( d[ dreg ].l );
}

// dn x d16(An) -> dn
if (smode == 5 ){
    displ = (signed short int)memory->GetWord( pc + 2 )+ a[ sreg ].l;
    d[ dreg ].l = d[ dreg ].w * ( unsigned short int )memory->GetWord( displ );
    SETNZL( d[ dreg ].l );
    pc += 4;
}

// dn x d8(An, Xn) -> dn
if ( smode == 6 ){
    displ = (signed char)memory->GetByte( pc + 3 )+ a[ sreg ].l
       + d[ GETBITS( pc + 2, 0xF000, 12 )].l;
    d[ dreg ].l = d[ dreg ].w * ( unsigned short int )memory->GetWord( displ );
    SETNZL( d[ dreg ].l );
    pc += 4;
}

// dn x (xxx).w -> dn
if ( smode == 7 && sreg == 0 ){
    displ = memory->GetWord( pc + 2 );
    d[ dreg ].l = d[ dreg ].w * ( unsigned short int )memory->GetWord( displ );
    SETNZL( d[ dreg ].l );
    pc += 4;
}

// dn x (xxx).L -> dn
if (smode == 7 && sreg == 1 ){
    displ = memory->GetLongword( pc + 2 );
    d[ dreg ].l = d[ dreg ]. w * ( unsigned short int )memory->GetWord( displ );
    SETNZL( d[ dreg ].l );
    pc += 6;
}

// dn x #<xxx> -> dn
if ( smode == 7 && sreg == 4 ){
    displ = memory->GetWord( pc + 2 );
    d[ dreg ].l = d[ dreg ].w * ( unsigned short int )displ;
    SETNZL( d[ dreg ].l );
    pc += 4;
}
```

```
    /*
    OMIT: d16(PC) and d8(An, Xn)
    */
}

void M68008::muluLong()
{
    // Invalid Size
}

void M68008::nbcd()
{
}

void M68008::neg()
{
    // stores the opcode in ch
    int ch = memory->GetWord( pc );
    // holder for the new pc
    long newpc = pc;

    //
    //
    // If mode is 000 then it is a data register that is being accessed
    if ( ( ch & 0x38) == 0) {

        // if the size is 0 then it is a byte operation
        if ( ( ch & 0xc0) == 0) {

            printf( "NEG instruction with data register.b\n" );

            int contents = (signed char)d[(ch & 0x7)].b;
            contents = 0 - contents;

            d[(ch & 0x7)].b = contents;


            // flags
            ccr.n = ((contents & 0x80) >> 7);
            ccr.z = !(contents & 0xff) ;
            ccr.x = ccr.c = !ccr.z;
            // if b is negative, and result is negative then overflow
            ccr.v = (contents == 128);

            newpc += 2;
        }
        // if the size is 1 then it is a word operation
        if ( ( ( ch & 0xc0) >> 6 )== 1) {

            printf("NEG instruction with data register.w\n" );
```

```
        int contents = (signed)d[(ch & 0x7)].w;
        int bneg = ((contents & 0x8000) >> 15);
        contents = 0 - contents;

        d[(ch & 0x7)].w = contents;

        // flags
        // flags
        ccr.n = ((contents & 0x8000) >> 15);
        ccr.z = !(contents & 0xffff) ;
        ccr.x = ccr.c = !ccr.z;
        if ( (bneg) && (ccr.n) ) {
           // if b is negative, and result is negative then overflow
           ccr.v = 1;
        }
        newpc += 2;
    }
    // if the size is 2 then it is a long operation
    if ( ( (ch & 0xc0) >> 6 )== 2) {

        printf("NEG_instruction_with_data_register.l\n" );

        int contents = (signed)d[(ch & 0x7)].l;
        int bneg = ((contents & 0x80000000) >> 31);
        contents = 0 - contents;

        d[(ch & 0x7)].l = contents;

        // flags
        ccr.n = ((contents & 0x80000000) >> 31);
        ccr.z = !(contents & 0 xffffffff ) ;
        ccr.x = ccr.c = !ccr.z;
        if ( (bneg) && (ccr.n) ) {
           // if b is negative, and result is negative then overflow
           ccr.v = 1;
        }
        newpc += 2;
    }
}

//
//
//
// If mode is 010 then it is a address register that is being accessed
if ( ( (ch & 0x38) >> 3 )== 2) {

    // if the size is 0 then it is a byte operation
    if ( (ch & 0xc0) == 0) {

        printf( "NEG_instruction_with_address_register.b\n" );

        int address = a[(ch & 0x7)].l;
```

```
    int contents = (signed char)memory->GetByte( address );
    contents = 0 - contents;

    memory->SetByte( contents, address );

    // flags
    ccr.n = ( ( contents & 0x80 ) >> 7);
    ccr.z = !( contents & 0xff ) ;
    ccr.x = ccr.c = !ccr.z;
    // if b is negative, and result is negative then overflow
    ccr.v = ( contents == 128 );

    newpc += 2;
}

// if the size is 1 then it is a word operation
if ( ( (  ch & 0xc0 ) >> 6 )== 1) {

    printf( "NEG instruction with address register.w\n" );


    int address = a[(ch & 0x7)].l;

    int contents = (signed)memory->GetWord( address );
    int bneg = ((contents & 0x8000) >> 15);
    contents = 0 - contents;

    memory->SetWord( contents, address );

    // flags
    ccr.n = ( ( contents & 0x8000 ) >> 15);
    ccr.z = !( contents & 0xffff ) ;
    ccr.x = ccr.c = !ccr.z;
    if ( (bneg) && (ccr.n) ) {
        // if b is negative, and result is negative then overflow
        ccr.v = 1;
    }

    newpc += 2;
}

// if the size is 2 then it is a long operation
if ( ( (  ch & 0xc0 ) >> 6 )== 2) {

    printf( "NEG instruction with address register.l\n" );


    int address = a[(ch & 0x7)].l;

    int contents = (signed)memory->GetLongword( address );
    int bneg = ((contents & 0x80000000) >> 31);
    contents = 0 - contents;
```

```
        memory->SetLongword( contents, address );

        // flags
        ccr.n = ( ( contents & 0x80000000 )>> 31);
        ccr.z = !( contents & 0 xffffffff  ) ;
        ccr.x = ccr.c = !ccr.z;
        if ( ( bneg) && (ccr.n) ) {
           // if b is negative, and result is negative then overflow
           ccr.v = 1;
        }

        newpc += 2;
    }
}

//
//
//
// If mode is 011 then it is a (address register)+ that is being accessed
if ( ( ( ch & 0x38) >> 3 )== 3) {

    // if the size is 0 then it is a byte operation
    if ( ( (  ch & 0xc0 ) >> 6 )== 0) {

        printf( "NEG instruction with (address register.b)+\n" );

        int address = a[(ch & 0x7)].l;


        int contents = (signed char)memory->GetByte( address );
        contents = 0 - contents;

        memory->SetByte( contents, address );

        // increment the address register.
        a[(ch & 0x7)].l = address + 1;

        // flags
        ccr.n = ( ( contents & 0x80 ) >> 7);
        ccr.z = !( contents & 0xff ) ;
        ccr.x = ccr.c = !ccr.z;
        // if b is negative, and result is negative then overflow
        ccr.v = ( contents == 128 );

        newpc += 2;
    }

    // if the size is 1 then it is a word operation
    if ( ( (  ch & 0xc0 ) >> 6 )== 1) {

        printf( "NEG instruction with (address register.w)+\n" );
```

```
        int address = a[(ch & 0x7)].l;

        int contents = (signed)memory->GetWord( address );
        int bneg = ((contents & 0x8000) >> 15);
        contents = 0 - contents;

        memory->SetWord( contents, address );

        // increment the address register.
        a[(ch & 0x7)].l = address + 2;

        // flags
        ccr.n = ( ( contents & 0x8000 ) >> 15);
        ccr.z = !( contents & 0xffff ) ;
        ccr.x = ccr.c = !ccr.z;
        if ( ( bneg) && (ccr.n) ) {
            // if b is negative, and result is negative then overflow
            ccr.v = 1;
        }

        newpc += 2;
    }

    // if the size is 2 then it is a long operation
    if ( ( ( ch & 0xc0 ) >> 6 )== 2) {

        printf( "NEG instruction with (address register.l)+\n" );

        int address = a[(ch & 0x7)].l;

        int contents = (signed)memory->GetLongword( address );
        int bneg = ((contents & 0x80000000) >> 31);
        contents = 0 - contents;

        memory->SetLongword( contents, address );

        // increment the address register.
        a[(ch & 0x7)].l = address + 4;

        // flags
        ccr.n = ( ( contents & 0x80000000 )>> 31);
        ccr.z = !( contents & 0 xffffffff ) ;
        ccr.x = ccr.c = !ccr.z;
        if ( ( bneg) && (ccr.n) ) {
            // if b is negative, and result is negative then overflow
            ccr.v = 1;
        }

        newpc += 2;
    }
}
```

```
//
//
//
// If mode is 100 then it is a -(address register) that is being accessed
if ( ( ( ch & 0x38) >> 3 )== 4) {

   // if the size is 0 then it is a byte operation
   if ( ( (  ch & 0xc0 ) >> 6 )== 0) {

      printf( "NEG instruction with -(address register.b)\n" );

      int address = a[(ch & 0x7)].l;

      // decrement the address register.
      address -= 1;
      a[(ch & 0x7)].l = address;

      int contents = (signed char)memory->GetByte( address );
      int bneg = ( (contents & 0x80) >> 7);
      contents = 0 - contents;

      memory->SetByte( contents, address );

      // flags
      ccr.n = ( ( contents & 0x80 ) >> 7);
      ccr.z = !( contents & 0xff ) ;
      ccr.x = ccr.c = !ccr.z;
      // if b is negative, and result is negative then overflow
      if ( ( bneg) && (ccr.n) ) {
         // if b is negative, and result is negative then overflow
         ccr.v = 1;
      }

      newpc += 2;
   }

   // if the size is 1 then it is a word operation
   if ( ( (  ch & 0xc0 ) >> 6 )== 1) {

      printf( "NEG instruction with (address register.w)+\n" );

      int address = a[(ch & 0x7)].l;

      // decrement the address register.
      address -= 2;
      a[(ch & 0x7)].l = address;

      int contents = (signed)memory->GetWord( address );
      int bneg = ((contents & 0x8000) >> 15);
      contents = 0 - contents;
```

```
        memory->SetWord( contents, address );

        // flags
        ccr.n = ( ( contents & 0x8000 ) >> 15);
        ccr.z = !( contents & 0xffff ) ;
        ccr.x = ccr.c = !ccr.z;
        if ( ( bneg) && (ccr.n) ) {
            // if b is negative, and result is negative then overflow
            ccr.v = 1;
        }

        newpc += 2;
    }

    // if the size is 2 then it is a long operation
    if ( ( (  ch & 0xc0 ) >> 6 )== 2) {

        printf( "NEG instruction with (address register.l)+\n" );

        int address = a[(ch & 0x7)].l;

        // decrement the address register.
        address -= 4;
        a[(ch & 0x7)].l = address;

        int contents = (signed)memory->GetLongword( address );
        int bneg = ((contents & 0x80000000) >> 31);
        contents = 0 - contents;

        memory->SetLongword( contents, address );

        // flags
        ccr.n = ( ( contents & 0x80000000 )>> 31);
        ccr.z = !( contents & 0 xffffffff ) ;
        ccr.x = ccr.c = !ccr.z;
        if ( ( bneg) && (ccr.n) ) {
            // if b is negative, and result is negative then overflow
            ccr.v = 1;
        }

        newpc += 2;
    }
}

//
//
//
// If mode is 111 then it is a (XXX) instruction
if ( ( ( ch & 0x38) >> 3 )== 7 ){
    // if register is 0 then it is in the form (XXX).W
    if ( (  ch & 0x7 ) == 0 ){
```

```
// if the size is 0 then it is a byte operation
if ( ( ( ch & 0xc0 ) >> 6 )== 0) {

    printf( "NEG instruction with (XXX.b).W\n" );

    short int address = memory->GetWord( pc + 2 );

    int contents = (signed char)memory->GetByte( address );
    short int bneg = ( (contents & 0x80) >> 7);
    contents = 0 - contents;

    memory->SetByte( contents, address );

    // flags
    ccr.n = ( ( contents & 0x80 ) >> 7);
    ccr.z = !( contents & 0xff ) ;
    ccr.x = ccr.c = !ccr.z;
    // if b is negative, and result is negative then overflow
    if ( ( bneg) && (ccr.n) ) {
        // if b is negative, and result is negative then overflow
        ccr.v = 1;
    }
}

// if the size is 1 then it is a word operation
if ( ( ( ch & 0xc0 ) >> 6 )== 1) {

    printf( "NEG instruction with (XXX.w).W\n" );

    short int address = memory->GetWord( pc + 2 );

    short int contents = (signed)memory->GetWord( address );
    short int bneg = ((contents & 0x8000) >> 15);
    contents = 0 - contents;

    memory->SetWord( contents, address );

    // flags
    ccr.n = ( ( contents & 0x8000 ) >> 15);
    ccr.z = !( contents & 0xffff ) ;
    ccr.x = ccr.c = !ccr.z;
    if ( ( bneg) && (ccr.n) ) {
        // if b is negative, and result is negative then overflow
        ccr.v = 1;
    }
}

// if the size is 2 then it is a long operation
if ( ( ( ch & 0xc0 ) >> 6 )== 2) {

    printf( "NEG instruction with (XXX.l).W\n" );
```

```
        int address = memory->GetWord( pc + 2 );

        int contents = (signed)memory->GetLongword( address );
        int bneg = ((contents & 0x80000000) >> 31);
        contents = 0 - contents;

        memory->SetLongword( contents, address );

        // flags
        ccr.n = ( ( contents & 0x80000000 )>> 31);
        ccr.z = !( contents & 0 xffffffff  ) ;
        ccr.x = ccr.c = !ccr.z;
        if ( ( bneg) && (ccr.n) ) {
           // if b is negative, and result is negative then overflow
           ccr.v = 1;
        }


    }
    newpc += 4;
}

/*
    OMIT
    THIS INSTRUCTION HAS NOT BEEN TESTED
    BECAUSE WE ARE UNABLE TO GET ROBODEV TO COMPILE IT

*/

// if register is 1 then it is in the form (XXX).L
if ( ( ch & 0x7 ) == 1 ){

    // if the size is 0 then it is a byte operation
    if ( ( ( ch & 0xc0 ) >> 6 )== 0) {

        printf( "NEG instruction with (XXX.b).L\n" );

        long int address = memory->GetLongword( pc + 2 );

        int contents = (signed char)memory->GetByte( address );
        short int bneg = ( (contents & 0x80) >> 7);
        contents = 0 - contents;

        memory->SetByte( contents, address );

        // flags
        ccr.n = ( ( contents & 0x80 ) >> 7);
        ccr.z = !( contents & 0xff ) ;
        ccr.x = ccr.c = !ccr.z;
        // if b is negative, and result is negative then overflow
        if ( ( bneg) && (ccr.n) ) {
           // if b is negative, and result is negative then overflow
```

```
        ccr.v = 1;
    }
}

// if the size is 1 then it is a word operation
if ( ( ( ch & 0xc0 ) >> 6 )== 1) {

    printf( "NEG instruction with (XXX.w).L\n" );

    long int address = memory->GetLongword( pc + 2 );

    short int contents = (signed)memory->GetWord( address );
    short int bneg = ((contents & 0x8000) >> 15);
    contents = 0 - contents;

    memory->SetWord( contents, address );

    // flags
    ccr.n = ( ( contents & 0x8000 ) >> 15);
    ccr.z = !( contents & 0xffff ) ;
    ccr.x = ccr.c = !ccr.z;
    if ( ( bneg) && (ccr.n) ) {
        // if b is negative, and result is negative then overflow
        ccr.v = 1;
    }
}

// if the size is 2 then it is a long operation
if ( ( ( ch & 0xc0 ) >> 6 )== 2) {

    printf( "NEG instruction with (XXX.l).L\n" );

    long int address = memory->GetLongword( pc + 2 );

    int contents = (signed)memory->GetLongword( address );
    int bneg = ((contents & 0x80000000) >> 31);
    contents = 0 - contents;

    memory->SetLongword( contents, address );

    // flags
    ccr.n = ( ( contents & 0x80000000 )>> 31);
    ccr.z = !( contents & 0 xffffffff ) ;
    ccr.x = ccr.c = !ccr.z;
    if ( ( bneg) && (ccr.n) ) {
        // if b is negative, and result is negative then overflow
        ccr.v = 1;
    }


}
newpc += 4;
```

```
        }
    }

    pc = newpc;
}

void M68008::negx()
{
    // stores the opcode in ch
    int ch = memory->GetWord( pc );
    // holder for the new pc
    long newpc = pc;

    //
    //
    // If mode is 000 then it is a data register that is being accessed
    if ( ( ch & 0x38) == 0) {

        // if the size is 0 then it is a byte operation
        if ( ( ch & 0xc0) == 0) {

            printf( "NEGX instruction with data register.b\n" );

            int contents = (signed char)d[(ch & 0x7)].b;
            int temp = contents + ccr.x;
            contents = 0 - temp;

            d[(ch & 0x7)].b = contents;


            // flags
            ccr.n = ((contents & 0x80) >> 7);
            ccr.x = ccr.c =  (temp > 0);
            if ((contents & 0xff) != 0) {
                ccr.z = 0;
            }
            // if b is negative, and result is negative then overflow
            ccr.v = (contents == 128);

            newpc += 2;
        }
        // if the size is 1 then it is a word operation
        if ( ( ( ch & 0xc0) >> 6 )== 1) {

            printf("NEGX instruction with data register.w\n" );

            int contents = (signed)d[(ch & 0x7)].w;
            int B = contents + ccr.x;
            int bneg = ((B & 0x8000) >> 15);
            contents = 0 - B;

            d[(ch & 0x7)].w = contents;
```

```
        // flags
        ccr.n = ((contents & 0x8000) >> 15);
        ccr.x = ccr.c =  (B > 0);
        if ((contents & 0xffff) != 0) {
            ccr.z = 0;
        }
        if ( (bneg) && (ccr.n) ) {
            // if b is negative, and result is negative then overflow
            ccr.v = 1;
        }

        newpc += 2;
    }
    // if the size is 2 then it is a long operation
    if ( ( (ch & 0xc0) >> 6 )== 2) {

        printf("NEGX_instruction_with_data_register.l\n" );

        int contents = (signed)d[(ch & 0x7)].l;
        int B = contents + ccr.x;
        int bneg = ((B & 0x80000000) >> 31);
        contents = 0 - B;

        d[(ch & 0x7)].l = contents;

        // flags
        ccr.n = ((contents & 0x80000000) >> 31);
        ccr.x = ccr.c =  (B > 0);
        if ((contents & 0 xffffffff ) != 0) {
            ccr.z = 0;
        }
        if ( (bneg) && (ccr.n) ) {
            // if b is negative, and result is negative then overflow
            ccr.v = 1;
        }

        newpc += 2;
    }
}

//
//
//
// If mode is 010 then it is a address register that is being accessed
if ( ( (ch & 0x38) >> 3 )== 2) {

    // if the size is 0 then it is a byte operation
    if ( (ch & 0xc0) == 0) {

        printf( "NEGX_instruction_with_address_register.b\n" );
```

```
    int address = a[(ch & 0x7)].l;
    int contents = (signed char)memory->GetByte( address );
    int B = contents + ccr.x;
    contents = 0 - B;

    memory->SetByte( contents, address );

    // flags
    ccr.n = ((contents & 0x80) >> 7);
    ccr.x = ccr.c = (B > 0);
    if ((contents & 0xff) != 0) {
        ccr.z = 0;
    }
    // if b is negative, and result is negative then overflow
    ccr.v = (contents == 0x80);

    newpc += 2;
}

// if the size is 1 then it is a word operation
if ( ( (  ch & 0xc0 ) >> 6 )== 1) {

    printf( "NEGX instruction with address register.w\n" );


    int address = a[(ch & 0x7)].l;

    int contents = (signed)memory->GetWord( address );
    int B = contents + ccr.x;
    int bneg = ((B & 0x8000) >> 15);
    contents = 0 - B;

    memory->SetWord( contents, address );

    // flags
    ccr.n = ( ( contents & 0x8000 ) >> 15);
    ccr.x = ccr.c = (B > 0);
    if ( ( bneg) && (ccr.n) ) {
        // if b is negative, and result is negative then overflow
        ccr.v = 1;
    }
    if ((contents & 0xffff ) != 0) {
        ccr.z = 0;
    }

    newpc += 2;
}

// if the size is 2 then it is a long operation
if ( ( (  ch & 0xc0 ) >> 6 )== 2) {

    printf( "NEGX instruction with address register.l\n" );
```

```
        int address = a[(ch & 0x7)].l;

        int contents = (signed)memory->GetLongword( address );
        int B = contents + ccr.x;
        int bneg = ((B & 0x80000000) >> 31);
        contents = 0 - B;

        memory->SetLongword( contents, address );

        // flags
        ccr.n = ( ( contents & 0x80000000 )>> 31);
        ccr.x = ccr.c = (B > 0);
        if ( (bneg) && (ccr.n) ) {
           // if b is negative, and result is negative then overflow
           ccr.v = 1;
        }
        if ((contents & 0 xffffffff ) != 0) {
           ccr.z = 0;
        }

        newpc += 2;
     }
}

//
//
//
// If mode is 011 then it is a (address register)+ that is being accessed
if ( ( ( ch & 0x38) >> 3 )== 3) {

   // if the size is 0 then it is a byte operation
   if ( ( (  ch & 0xc0 ) >> 6 )== 0) {

      printf( "NEGX instruction with (address register.b)+\n" );

      int address = a[(ch & 0x7)].l;


      int contents = (signed char)memory->GetByte( address );
      int B = contents - ccr.x;
      contents = 0 - B;

      memory->SetByte( contents, address );

      // increment the address register.
      a[(ch & 0x7)].l = address + 1;

      // flags
      ccr.n = ( ( contents & 0x80 ) >> 7);
      ccr.x = ccr.c = (B > 0);
```

```
    // if b is negative, and result is negative then overflow
    ccr.v = ( contents == 128 );
    if ((contents & 0xff) != 0) {
        ccr.z = 0;
    }

    newpc += 2;
}

// if the size is 1 then it is a word operation
if ( ( (  ch & 0xc0 ) >> 6 )== 1) {

    printf( "NEGX instruction with (address register.w)+\n" );

    int address = a[(ch & 0x7)].l;

    int contents = (signed)memory->GetWord( address );
    int B = contents + ccr.x;
    int bneg = ((B & 0x8000) >> 15);
    contents = 0 - B;

    memory->SetWord( contents, address );

    // increment the address register.
    a[(ch & 0x7)].l = address + 2;

    // flags
    ccr.n = ( ( contents & 0x8000 ) >> 15);
    ccr.x = ccr.c = (B > 0);
    if ( (bneg) && (ccr.n) ) {
        // if b is negative, and result is negative then overflow
        ccr.v = 1;
    }
    if ((contents & 0xffff ) != 0) {
        ccr.z = 0;
    }

    newpc += 2;
}

// if the size is 2 then it is a long operation
if ( ( (  ch & 0xc0 ) >> 6 )== 2) {

    printf( "NEGX instruction with (address register.l)+\n" );

    int address = a[(ch & 0x7)].l;

    int contents = (signed)memory->GetLongword( address );
    int B = contents + ccr.x;
    int bneg = ((B & 0x80000000) >> 31);
    contents = 0 - B;
```

```
        memory->SetLongword( contents, address );

        // increment the address register.
        a[(ch & 0x7)].l = address + 4;

        // flags
        ccr.n = ( ( contents & 0x80000000 )>> 31);
        ccr.x = ccr.c = (B > 0);
        if ( (bneg) && (ccr.n) ) {
           // if b is negative, and result is negative then overflow
           ccr.v = 1;
        }
        if ((contents & 0 xffffffff ) != 0) {
           ccr.z = 0;
        }

        newpc += 2;
     }
  }

  //
  //
  //
  // If mode is 100 then it is a -(address register) that is being accessed
  if ( ( ( ch & 0x38) >> 3 )== 4) {

     // if the size is 0 then it is a byte operation
     if ( ( (  ch & 0xc0 ) >> 6 )== 0) {

        printf( "NEGX instruction with -(address register.b)\n" );

        int address = a[(ch & 0x7)].l;

        // decrement the address register.
        address -= 1;
        a[(ch & 0x7)].l = address;

        int contents = (signed char)memory->GetByte( address );
        int B = contents - ccr.x;
        int bneg = ( (B & 0x80) >> 7);
        contents = 0 - B;

        memory->SetByte( contents, address );

        // flags
        ccr.n = ( ( contents & 0x80 ) >> 7);
        ccr.x = ccr.c = (B > 0);
        // if b is negative, and result is negative then overflow
        if ( (bneg) && (ccr.n) ) {
           // if b is negative, and result is negative then overflow
           ccr.v = 1;
        }
```

```
    if ((contents & 0xff) != 0) {
        ccr.z = 0;
    }

    newpc += 2;
}

// if the size is 1 then it is a word operation
if ( ( (  ch & 0xc0 ) >> 6 )== 1) {

    printf( "NEGX instruction with (address register.w)+\n" );

    int address = a[(ch & 0x7)].l;

    // decrement the address register.
    address -= 2;
    a[(ch & 0x7)].l = address;

    int contents = (signed)memory->GetWord( address );
    int B = contents - ccr.x;
    int bneg = ((B & 0x8000) >> 15);
    contents = 0 - B;

    memory->SetWord( contents, address );

    // flags
    ccr.n = ( ( contents & 0x8000 ) >> 15);
    ccr.x = ccr.c = (B > 0);
    if ( ( bneg) && (ccr.n) ) {
        // if b is negative, and result is negative then overflow
        ccr.v = 1;
    }
    if ((contents & 0xffff ) != 0) {
        ccr.z = 0;
    }

    newpc += 2;
}

// if the size is 2 then it is a long operation
if ( ( (  ch & 0xc0 ) >> 6 )== 2) {

    printf( "NEGX instruction with (address register.l)+\n" );

    int address = a[(ch & 0x7)].l;

    // decrement the address register.
    address -= 4;
    a[(ch & 0x7)].l = address;

    int contents = (signed)memory->GetLongword( address );
    int B = contents + ccr.x;
```

```
        int bneg = ((B & 0x80000000) >> 31);
        contents = 0 - B;

        memory->SetLongword( contents, address );

        // flags
        ccr.n = ( ( contents & 0x80000000 )>> 31);
        ccr.x = ccr.c = (B > 0);
        if ( ( bneg) && (ccr.n) ) {
           // if b is negative, and result is negative then overflow
           ccr.v = 1;
        }
        if ((contents & 0 xffffffff ) != 0) {
           ccr.z = 0;
        }

        newpc += 2;
     }
  }

  //
  //
  //
  // If mode is 111 then it is a (XXX) instruction
  if ( ( ( ch & 0x38) >> 3 )== 7 ){
     // if register is 0 then it is in the form (XXX).W
     if ( ( ch & 0x7 ) == 0 ){

        // if the size is 0 then it is a byte operation
        if ( ( ( ch & 0xc0 ) >> 6 )== 0) {

           printf( "NEGX instruction with (XXX.b).W\n" );

           short int address = memory->GetWord( pc + 2 );

           int contents = (signed char)memory->GetByte( address );
           int B = contents + ccr.x;
           short int bneg = ( (B & 0x80) >> 7);
           contents = 0 - B;

           memory->SetByte( contents, address );

           // flags
           ccr.n = ( ( contents & 0x80 ) >> 7);
           ccr.x = ccr.c = (B > 0);
           // if b is negative, and result is negative then overflow
           if ( ( bneg) && (ccr.n) ) {
              // if b is negative, and result is negative then overflow
              ccr.v = 1;
           }
           if ((contents & 0xff) != 0) {
              ccr.z = 0;
```

```
    }
}

// if the size is 1 then it is a word operation
if ( ( ( ch & 0xc0 ) >> 6 )== 1) {

    printf( "NEGX instruction with (XXX.w).W\n" );

    short int address = memory->GetWord( pc + 2 );

    short int contents = (signed)memory->GetWord( address );
    int B = contents + ccr.x;
    short int bneg = ((B & 0x8000) >> 15);
    contents = 0 - B;

    memory->SetWord( contents, address );

    // flags
    ccr.n = ( ( contents & 0x8000 ) >> 15);
    ccr.x = ccr.c = (B > 0);
    if ( ( bneg) && (ccr.n) ) {
        // if b is negative, and result is negative then overflow
        ccr.v = 1;
    }
    if ((contents & 0xffff ) != 0) {
        ccr.z = 0;
    }
}

// if the size is 2 then it is a long operation
if ( ( ( ch & 0xc0 ) >> 6 )== 2) {

    printf( "NEGX instruction with (XXX.l).W\n" );

    int address = memory->GetWord( pc + 2 );

    int contents = (signed)memory->GetLongword( address );
    int B = contents + ccr.x;
    int bneg = ((B & 0x80000000) >> 31);
    contents = 0 - B;

    memory->SetLongword( contents, address );

    // flags
    ccr.n = ( ( contents & 0x80000000 )>> 31);
    ccr.x = ccr.c = (B > 0);
    if ( ( bneg) && (ccr.n) ) {
        // if b is negative, and result is negative then overflow
        ccr.v = 1;
    }
    if ((contents & 0 xffffffff ) != 0) {
        ccr.z = 0;
```

```
            }


         }
         newpc += 4;
      }

      /*
         OMIT
         THIS INSTRUCTION HAS NOT BEEN TESTED
         BECAUSE WE ARE UNABLE TO GET ROBODEV TO COMPILE IT

      */


   }

   pc = newpc;
}

void M68008::nop()
{
   pc += 2;
}

/* OMITTED is the displacement to the address register operand to this instruction. */
void M68008::not() {
   long int newpc = pc;

   // short int instruction = memory->GetWord( pc );

   // if the mode is a word address
   if ( ( GETBITS( pc, 56, 0) == 56 )){

      // if the register is 000, it is a word address.
      if ((GETBITS( pc, 7, 0) == 0 )){
         printf("not with word operand.\n");

         if (GETBITS( pc, 192, 6) == 0) {
            // byte operation
            // get memory location
            short int operand = memory->GetWord( pc + 2);
            // get the value stored at the location
            unsigned int contents = memory->GetWord( operand )& 0xff00;
            contents = contents >> 8;
            // not the value at the location
            contents = ~contents;
            contents = contents & 0xFF;

            // store it back in memory at the location
            memory->SetByte( contents, operand );
```

```
    // setting flags
    if ( contents >= 0x80) {
        ccr.n = 1;
    } else {
        ccr.n = 0;
    }

    if ( contents == 0 ){
        ccr.z = 1;
    } else {
        ccr.z = 0;
    }

    ccr.v = 0;
    ccr.c = 0;
}

if (GETBITS( pc, 192, 6) == 1) {
    // word operation
    // get memory location
    short int operand = memory->GetWord( pc + 2);
    // get the value stored at the location
    unsigned int contents = memory->GetWord( operand );
    contents = contents & 0xffff;
    // not the value at the location
    contents = ~contents;
    contents = contents & 0xffff;
    // store it back in memory at the location
    memory->SetWord( contents, operand );

    // setting flags
    if ( contents >= 0x8000 ){
        ccr.n = 1;
    } else {
        ccr.n = 0;
    }

    if ( contents == 0 ){
        ccr.z = 1;
    } else {
        ccr.z = 0;
    }

    ccr.v = 0;
    ccr.c = 0;
}

if (GETBITS( pc, 192, 6) == 2) {
    // long operation
    // get memory location
    short int operand = memory->GetWord( pc + 2);
    // get the value stored at the location
```

```
        long int contents = memory->GetLongword( operand );

        // not the value at the location
        contents = ~contents;
        // store it back in memory at the location
        memory->SetLongword( contents, operand );

        // setting flags
        if ( GETLONGBITS( operand, 0x80000000, 31 )== 1 ){
            ccr.n = 1;
        } else {
            ccr.n = 0;
        }

        if ( GETLONGBITS( operand, 0xffffffff, 0 )== 0 ){
            ccr.z = 1;
        } else {
            ccr.z = 0;
        }

        ccr.v = 0;
        ccr.c = 0;
    }

    newpc += 4;
}

//
//
// if the register is 001, it is a long address.
if ((GETBITS( pc, 7, 0) == 1 )){
    printf("not_with_long_operand.\n");

    if (GETBITS( pc, 192, 6) == 0) {
        // byte operation
        // get memory location
        long int operand = memory->GetLongword( pc + 2);
        // get the value stored at the location
        short int contents = (int)memory->GetByte( operand );
        // not the value at the location
        contents = ~contents;
        // store it back in memory at the location
        memory->SetByte( (char)contents, operand );

        // setting flags
        if (GETBITS( operand, 0x80, 7) == 1) {
            ccr.n = 1;
        } else {
            ccr.n = 0;
        }

        if (GETBITS( operand, 0xff, 0 )== 0 ){
```

```
            ccr.z = 1;
        } else {
            ccr.z = 0;
        }

        ccr.v = 0;
        ccr.c = 0;
    }

    if (GETBITS( pc, 192, 6) == 1) {
        // word operation
        // get memory location
        long int operand = memory->GetLongword( pc + 2);
        // get the value stored at the location
        short int contents = memory->GetWord( operand );
        // not the value at the location
        contents = ~contents;
        // store it back in memory at the location
        memory->SetWord( contents, operand );

        // setting flags
        if (GETBITS( operand, 0x8000, 15) == 1) {
            ccr.n = 1;
        } else {
            ccr.n = 0;
        }

        if (GETBITS( operand, 0xffff, 0 )== 0 ){
            ccr.z = 1;
        } else {
            ccr.z = 0;
        }

        ccr.v = 0;
        ccr.c = 0;
    }

    if (GETBITS( pc, 192, 6) == 2) {
        // long operation
        // get memory location
        long int operand = memory->GetLongword( pc + 2);
        // get the value stored at the location
        long int contents = memory->GetLongword( operand );

        // not the value at the location
        contents = ~contents;
        // store it back in memory at the location
        memory->SetLongword( contents, operand );

        // setting flags
        if (GETLONGBITS( operand, 0x80000000, 31) == 1) {
            ccr.n = 1;
```

```
            } else {
                ccr.n = 0;
            }

            if (GETLONGBITS( operand, 0xffffffff, 0 )== 0 ){
                ccr.z = 1;
            } else {
                ccr.z = 0;
            }

            ccr.v = 0;
            ccr.c = 0;
        }
        newpc+= 6;
    }

}

//
//
// if the mode is a data register
if ( (GETBITS( pc, 56, 0) == 0 )){
    // decode the register number
    int var = GETBITS( pc, 7, 0);

    if (GETBITS( pc, 192, 6) == 0) {
        // byte op
        printf("This is a byte, not, data register op\n");
        // get the contents of the data register
        int contents = d[var].b;
        // not the value stored in the data register
        contents = ~contents;
        // store it back in the data register
        d[var].b = contents;

        // setting flags
        if (( (contents & 0x80) >> 7) == 1) {
            ccr.n = 1;
        } else {
            ccr.n = 0;
        }
        if (( contents & 0xff ) == 0 ){
            ccr.z = 1;
        } else {
            ccr.z = 0;
        }

        ccr.v = 0;
        ccr.c = 0;
    }

    if (GETBITS( pc, 192, 6) == 1) {
```

```
    // word op
    printf("This is a word, not, data register op\n");
    // get the contents of the data register
    int contents = d[var].w;
    // not the value stored in the data register
    contents = ~contents;
    // store it back in the data register
    d[var].w = contents;

    // setting flags
    if (( (contents & 0x8000) >> 15) == 1) {
        ccr.n = 1;
    } else {
        ccr.n = 0;
    }

    if (( contents & 0xffff ) == 0 ){
        ccr.z = 1;
    } else {
        ccr.z = 0;
    }

    ccr.v = 0;
    ccr.c = 0;
}


if (GETBITS( pc, 192, 6) == 2) {
    // long op
    printf("This is a long, not, data register op\n");
    // get the contents of the data register
    int contents = d[var].l;
    // not the value stored in the data register
    contents = ~contents;
    // store it back in the data register
    d[var].l = contents;

    // setting flags
    if ( (contents & 0x80000000) >> 31 == 1) {
        ccr.n = 1;
    } else {
        ccr.n = 0;
    }

    if (( contents & 0 xffffffff ) == 0 ){
        ccr.z = 1;
    } else {
        ccr.z = 0;
    }

    ccr.v = 0;
    ccr.c = 0;
```

```
    }
    newpc += 2;
}

//
//
// if the mode is an address register pointer
if ( ( GETBITS( pc, 56, 3) == 2 )){

    // decode the register number
    int var = GETBITS( pc, 7, 0);
    int address = a[var].l;

    if (GETBITS( pc, 192, 6) == 0) {
        // byte operation
        // get the contents to be notted
        signed int contents = memory->GetByte( address );
        // not the contents
        contents = ~contents;
        // store it back in memory
        memory->SetByte( contents, address );

        // setting flags
        if (((contents & 0x80) >> 7) == 1) {
            ccr.n = 1;
        } else {
            ccr.n = 0;
        }

        if (((  contents & 0xff) >> 0 )== 0 ){
            ccr.z = 1;
        } else {
            ccr.z = 0;
        }

        ccr.v = 0;
        ccr.c = 0;

    }


    if (GETBITS( pc, 192, 6) == 1) {
        // word operation
        // get the contents to be notted
        int contents = memory->GetWord( address );
        // not the contents
        contents = ~contents;
        // store it back in memory
        memory->SetWord( contents, address );

        // setting flags
        if (((contents & 0x8000) >> 15) == 1) {
```

```
            ccr.n = 1;
        } else {
            ccr.n = 0;
        }

        if ((contents & 0xffff) == 0 ){
            ccr.z = 1;
        } else {
            ccr.z = 0;
        }

        ccr.v = 0;
        ccr.c = 0;
    }


    if (GETBITS( pc, 192, 6) == 2) {
        // longword operation
        // get the contents to be notted
        long int contents = memory->GetLongword( address );
        // not the contents
        contents = ~contents;
        // store it back in memory
        memory->SetLongword( contents, address );


        // setting flags
        if (((contents & 0x80000000) >> 31) == 1) {
            ccr.n = 1;
        } else {
            ccr.n = 0;
        }

        if ((contents & 0 xffffffff ) == 0 ){
            ccr.z = 1;
        } else {
            ccr.z = 0;
        }

        ccr.v = 0;
        ccr.c = 0;
    }

    newpc += 2;
}

//
//
// if the mode is an post-increment address register pointer
if ( (GETBITS( pc, 56, 3) == 3 )){

    // decode the register number
```

```
int var = GETBITS( pc, 7, 0);
int address = a[var].l;

if (GETBITS( pc, 192, 6) == 0) {
   // byte operation
   // get the contents to be notted
   int contents = (int) memory->GetByte( address );
   // not the contents
   contents = ~contents;
   // store it back in memory
   memory->SetByte( (char) contents, address );

   // increment address by a byte value
   address += 1;

   // setting flags
   if (((contents & 0x80) >> 7) == 1) {
      ccr.n = 1;
   } else {
      ccr.n = 0;
   }

   if ((contents & 0xff) == 0 ){
      ccr.z = 1;
   } else {
      ccr.z = 0;
   }

   ccr.v = 0;
   ccr.c = 0;
}


if (GETBITS( pc, 192, 6) == 1) {
   // word operation
   // get the contents to be notted
   int contents = memory->GetWord( address );
   // not the contents
   contents = ~contents;
   // store it back in memory
   memory->SetWord( contents, address );

   // increment address by a word value
   address += 2;

   // setting flags
   if (((contents & 0x8000) >> 15) == 1) {
      ccr.n = 1;
   } else {
      ccr.n = 0;
   }
```

```
        if ((contents & 0xffff) == 0 ){
            ccr.z = 1;
        } else {
            ccr.z = 0;
        }

        ccr.v = 0;
        ccr.c = 0;
    }


    if (GETBITS( pc, 192, 6) == 2) {
        // longword operation
        // get the contents to be notted
        long int contents = memory->GetLongword( address );
        // not the contents
        contents = ~contents;
        // store it back in memory
        memory->SetLongword( contents, address );

        // increment address by a longword value
        address += 4;

        // setting flags
        if (((contents & 0x80000000) >> 31) == 1) {
            ccr.n = 1;
        } else {
            ccr.n = 0;
        }

        if ((contents & 0 xffffffff ) == 0 ){
            ccr.z = 1;
        } else {
            ccr.z = 0;
        }

        ccr.v = 0;
        ccr.c = 0;
    }

    a[var].l = address;
    newpc += 2;
}

    // if the mode is an pre-decrement address register pointer
if ( ( GETBITS( pc, 56, 3) == 4 )){

    // decode the register number
    int var = GETBITS( pc, 7, 0);
    int address = a[var].l;

    if (GETBITS( pc, 192, 6) == 0) {
```

```
    // decrement address by a byte value
    address -= 1;
    // byte operation
    // get the contents to be notted
    int contents = (int) memory->GetByte( address );
    // not the contents
    contents = ~contents;

    // store it back in memory
    memory->SetByte( (char) contents, address );

    // setting flags
    if (((contents & 0x80) >> 7) == 1) {
        ccr.n = 1;
    } else {
        ccr.n = 0;
    }

    if ((contents & 0xff) == 0 ){
        ccr.z = 1;
    } else {
        ccr.z = 0;
    }

    ccr.v = 0;
    ccr.c = 0;
}


if (GETBITS( pc, 192, 6) == 1) {
    // decrement address by a word value
    address -= 2;
    // word operation
    // get the contents to be notted
    int contents = memory->GetWord( address );
    // not the contents
    contents = ~contents;
    // store it back in memory
    memory->SetWord( contents, address );
    // setting flags
    if (((contents & 0x8000) >> 15) == 1) {
        ccr.n = 1;
    } else {
        ccr.n = 0;
    }

    if ((contents & 0xffff ) == 0 ){
        ccr.z = 1;
    } else {
        ccr.z = 0;
    }
```

```
            ccr.v = 0;
            ccr.c = 0;
        }


        if (GETBITS( pc, 192, 6) == 2) {
            // decrement address by a longword value
            address -= 4;
            // longword operation
            // get the contents to be notted
            long int contents = memory->GetLongword( address );
            // not the contents
            contents = ~contents;
            // store it back in memory
            memory->SetLongword( contents, address );

            // setting flags
            if (((contents & 0x80000000) >> 31) == 1) {
                ccr.n = 1;
            } else {
                ccr.n = 0;
            }

            if ((contents & 0 xffffffff ) == 0 ){
                ccr.z = 1;
            } else {
                ccr.z = 0;
            }

            ccr.v = 0;
            ccr.c = 0;
        }
        a[var].l = address;
        newpc += 2;
    }
    pc = newpc;
}

/* OMIT
    (d16, An) and (d16, An, Xn) modes have been omitted awaiting further information
*/
```

1.7 TInstrD-N.cpp

The translation methods of the instructions starting with $D \cdots N$.

```
/*
    Motorola 68008 Simulator
    (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
    Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

    This class represents the Motorola 68008 Microprocessor
```

```
   Instruction translation D-N
*/
#include <stdio.h>
#include "M68008.h"
#define GETBITS( addr, mask, shift )( memory->GetWord( addr )& mask )>> shift
#define GETLONGBITS( addr, mask, shift )( memory->GetLongword( addr )& mask )>> shift
#define SETNZ( src ){ ccr.n = ( (signed char)src < 0 ); ccr.z = ( src == 0 ); }
#define SETNZL( src ){ ccr.n = ( (signed long)src < 0 ); ccr.z = ( src == 0 ); }
#define SETNZW( src ){ ccr.n = ( (signed short)src < 0 ); ccr.z = ( src == 0 ); }

unsigned long M68008::tS_dbxx( unsigned long _address, char *buf )
{
   short int reg = GETBITS( _address, 0x0007, 0);
   short int condition = GETBITS( _address, 0x0f00, 8);
   signed short int displ = memory->GetWord( _address + 2 );

      switch( condition ) {
      case 0:      // True (not defined)
            sprintf ( buf, "dbt d%X,$%X", reg, displ );
            break;
      case 1:      // False (not defined)
            sprintf ( buf, "dbf d%X,$%X", reg, displ );
            break;
      case 2:      // BHI (HI = !C & !Z)
          sprintf ( buf, "dbhi d%X,$%X", reg, displ );
          break;
      case 3:      // BLS (LS = C | V)
          sprintf ( buf, "dbls d%X,$%X", reg, displ );
          break;
      case 4:      // BCC (CC = !C)
          sprintf ( buf, "dbcc d%X,$%X", reg, displ );
          break;
      case 5:      // BCS (CS = C)
          sprintf ( buf, "dbcs d%X,$%X", reg, displ );
          break;
      case 6:      // BNE (NE = !Z)
          sprintf ( buf, "dbne d%X,$%X", reg, displ );
          break;
      case 7:      // BEQ (EQ = Z)
          sprintf ( buf, "dbeq d%X,$%X", reg, displ );
          break;
      case 8:      // BVC (VC = !V)
          sprintf ( buf, "dbvc d%X,$%X", reg, displ );
          break;
      case 9:      // BVS (VS = V)
          sprintf ( buf, "dbvs d%X,$%X", reg, displ );
          break;
      case 10: // BPL (PL = !N)
          sprintf ( buf, "dbpl d%X,$%X", reg, displ );
          break;
      case 11: // BMI (MI = N)
          sprintf ( buf, "dbmi d%X,$%X", reg, displ );
```

```
            break;
        case 12: // BGE (GE = N & V | !N & !V)
            sprintf ( buf, "dbge d%X,$%X", reg, displ );
            break;
        case 13: // BLT (LT = N & !V | !N & V)
            sprintf ( buf, "dblt d%X,$%X", reg, displ );
            break;
        case 14: // BGT (GT = N & V & !Z | !N & !V & !Z)
            sprintf ( buf, "dbgt d%X,$%X", reg, displ );
            break;
        case 15: // BLE (LE = Z | N & !V | !N & V)
            sprintf ( buf, "dble d%X,$%X", reg, displ );
            break;
    }

    _address += 4;
    return _address;
}

unsigned long M68008::tS_divsWord( unsigned long _address, char *buf )
{
    short int sreg,dreg,smode;
    signed long displ;
    sreg = GETBITS( _address, 0x7, 0 );
    smode = GETBITS( _address, 0x38, 3 );
    dreg = GETBITS( _address, 0xE00, 9 );

    // dn / dn -> dn
    if ( smode == 0 ){
        sprintf ( buf, "divs.w d%X,d%X", sreg, dreg );
    }

    // dn / (An) -> dn
    if ( smode == 2 ){
        sprintf ( buf, "divs.w (a%X),d%X", sreg, dreg );
    }

    // dn / (An)+ -> dn
    if ( smode == 3 ){
        sprintf ( buf, "divs.w (a%X)+,d%X", sreg, dreg );
    }

    // dn / -(An) -> dn
    if ( smode == 4 ){
        sprintf ( buf, "divs.w -(a%X),d%X", sreg, dreg );
    }

    // dn /  d16(an)
    if ( smode == 5 )
    {
        displ = (signed short int)memory->GetWord( _address + 2 );
        if( decImm )sprintf ( buf, "divs.w %ld(a%X),d%X", displ, sreg, dreg );
```

```
        else sprintf( buf, "divs.w $%lX(a%X),d%X", displ, sreg, dreg );
        _address += 2;
    }

    // dn / d8(an, xn)
    if ( smode == 6 )
    {
        displ = (signed char)memory->GetByte( _address + 3 );
        int var = d[ GETBITS( _address + 2, 0xF000, 12 )].l;
        if( decImm )sprintf( buf, "divs.w %ld(a%X,d%X),d%X", displ, sreg, var, dreg );
        else sprintf( buf, "divs.w $%lX(a%X,d%X),d%X", displ, sreg, var, dreg );
        _address += 2;
    }

    if ( smode == 7 )
    {
        if ( sreg == 0 ){
            // dn / (XXX).W
            displ = memory->GetWord( _address + 2 );
            sprintf( buf, "divs.w $%lX,d%X", displ, dreg );
            _address += 2;
        }
        if ( sreg == 1 ){
            // dn / (XXX).l
            displ = memory->GetLongword( _address + 2 );
            sprintf( buf, "divs.w $%lX,d%X", displ, dreg );
            _address += 4;
        }

        if ( sreg == 4 )
        {
            displ = memory->GetWord( _address + 2 );
            if( decImm )sprintf( buf, "divs.w #%ld,d%X", displ, dreg );
            else sprintf( buf, "divs.w #$%lX,d%X", displ, dreg );
            _address += 2;
        }
    }
    _address += 2;
    return _address;
}

unsigned long M68008::tS_divsLong( unsigned long _address, char *buf )
{

    // invalid size code
    sprintf( buf, "divs.l: Unimplemented" );
    return _address;
}

unsigned long M68008::tS_divuWord( unsigned long _address, char *buf )
{
    short int sreg,dreg,smode;
```

```
unsigned long displ;
sreg = GETBITS( _address, 0x7, 0 );
smode = GETBITS( _address, 0x38, 3 );
dreg = GETBITS( _address, 0xE00, 9 );

// dn / dn -> dn
if ( smode == 0 ){
    sprintf ( buf, "divu.w␣d%X,d%X", sreg, dreg );
}

// dn / (An) -> dn
if ( smode == 2 ){
    sprintf ( buf, "divu.w␣(a%X),d%X", sreg, dreg );
}

// dn / (An)+ -> dn
if ( smode == 3 ){
    sprintf ( buf, "divu.w␣(a%X)+,d%X", sreg, dreg );
}

// dn / -(An) -> dn
if ( smode == 4 ){
    sprintf ( buf, "divu.w␣-(a%X),d%X", sreg, dreg );
}

// dn /  d16(an)
if ( smode == 5 )
{
    displ = (signed short int)memory->GetWord( _address + 2 );
    if( decImm )sprintf ( buf, "divu.w␣%ld(a%X),d%X", displ, sreg, dreg );
    else sprintf ( buf, "divu.w␣$%lX(a%X),d%X", displ, sreg, dreg );
    _address += 2;
}

// dn / d8(an, xn)
if ( smode == 6 )
{
    displ = (signed char)memory->GetByte( _address + 3 );
    int var = d[ GETBITS( _address + 2, 0xF000, 12 )].l;
    if( decImm )sprintf ( buf, "divu.w␣%ld(a%X,d%X),d%X", displ, sreg, var, dreg );
    else sprintf ( buf, "divu.w␣$%lX(a%X,d%X),d%X", displ, sreg, var, dreg );
    _address += 2;
}

if ( smode == 7 )
{
    if ( sreg == 0 ){
        // dn / (XXX).W
        displ = memory->GetWord( _address + 2 );
        sprintf ( buf, "divu.w␣$%lX,d%X", displ, dreg );
        _address += 2;
    }
```

```
        if ( sreg == 1 ){
            // dn / (XXX).l
            displ = memory->GetLongword( _address + 2 );
            sprintf ( buf, "divu.w $%lX,d%X", displ, dreg );
            _address += 4;
        }

        if ( sreg == 4 )
        {
            displ = memory->GetWord( _address + 2 );
            if ( decImm )sprintf ( buf, "divu.w #%ld,d%X", displ, dreg );
            else sprintf ( buf, "divu.w #$%lX,d%X", displ, dreg );
            _address += 2;
        }
    }
    _address += 2;

    return _address;
}

unsigned long M68008::tS_divuLong( unsigned long _address, char *buf )
{
    sprintf ( buf, "divu.l: Unimplemented" );
    //invalid size code

    return _address;
}

unsigned long M68008::tS_eor( unsigned long _address, char *buf )
{

    int reg, opmode, mode, eaReg;
    unsigned long displ;

    reg    = GETBITS( _address, 0x0E00, 9 );
    opmode = GETBITS( _address, 0x01C0, 6 );
    mode   = GETBITS( _address, 0x0038, 3 );
    eaReg  = GETBITS( _address, 0x0007, 0 );

    // dn,Dn
    if ( mode == 0 )
    {
        // byte
        if ( opmode == 4 )
        {
            sprintf ( buf, "eor.b d%X,d%X", reg, eaReg );
        }

        // word
        if ( opmode == 5 )
        {
            sprintf ( buf, "eor.w d%X,d%X", reg, eaReg );
```

```
    }

    // long
    if ( opmode == 6 )
    {
        sprintf ( buf, "eor.l d%X,d%X", reg, eaReg );
    }

    address += 2;
}

// (an),dn
if ( mode == 2 )
{
    // byte
    if ( opmode == 4 )
    {
        sprintf ( buf, "eor.b (a%X),d%X", reg, eaReg );
    }

    // word
    if ( opmode == 5 )
    {
        sprintf ( buf, "eor.w (a%X),d%X", reg, eaReg );
    }

    // long
    if ( opmode == 6 )
    {
        sprintf ( buf, "eor.l (a%X),d%X", reg, eaReg );
    }

    address += 2;
}

// (an)+,dn
if ( mode == 3 )
{
    // byte
    if ( opmode == 4 )
    {
        sprintf ( buf, "eor.b (a%X)+,d%X", reg, eaReg );
    }

    // word
    if ( opmode == 5 )
    {
        sprintf ( buf, "eor.w (a%X)+,d%X", reg, eaReg );
    }

    // long
    if ( opmode == 6 )
```

```
    {
        sprintf( buf, "eor.l (a%X)+,d%X", reg, eaReg );
    }

    _address += 2;
}

// -(an),dn
if( mode== 4 )
{

    // byte
    if( opmode == 4 )
    {
        sprintf( buf, "eor.b -(a%X),d%X", reg, eaReg );
    }

    // word
    if( opmode == 5 )
    {
        sprintf( buf, "eor.w -(a%X),d%X", reg, eaReg );
    }

    // long
    if( opmode == 6 )
    {
        sprintf( buf, "eor.l -(a%X),d%X", reg, eaReg );
    }

    _address += 2;
}

// d16(an),dn
if( mode == 5 )
{
    displ = ( signed short int )memory->GetWord( _address + 2 );

    // byte
    if( opmode == 4 )
    {
        if( decImm )sprintf( buf, "eor.b %ld(a%X),d%X", displ, reg, eaReg );
        else sprintf( buf, "eor.b $%lX(a%X),d%X", displ, reg, eaReg );
    }

    // word
    if( opmode == 5 )
    {
        if( decImm )sprintf( buf, "eor.w %ld(a%X),d%X", displ, reg, eaReg );
        else sprintf( buf, "eor.w $%lX(a%X),d%X", displ, reg, eaReg );
    }

    // long
```

```
    if( opmode == 6 )
    {
        if( decImm )sprintf( buf, "eor.l %ld(a%X),d%X", displ, reg, eaReg );
        else sprintf( buf, "eor.l $%lX(a%X),d%X", displ, reg, eaReg );
    }

    _address += 4;
}

// d8(an,xn),dn
if( mode == 6 )
{
    displ = (signed char)memory->GetByte( _address + 3 )& 0x0fff;
    int Xn = (signed char)memory->GetByte( _address + 3 )>> 12;

    // byte
    if( opmode == 4 )
    {
        if( decImm )sprintf( buf, "eor.b %ld(a%X,d%X),d%X", displ, reg, Xn, eaReg );
        else sprintf( buf, "eor.b $%lX(a%X,d%X),d%X", displ, reg, Xn, eaReg );
    }

    // word
    if( opmode == 5 )
    {
        if( decImm )sprintf( buf, "eor.w %ld(a%X,d%X),d%X", displ, reg, Xn, eaReg );
        else sprintf( buf, "eor.w $%lX(a%X,d%X),d%X", displ, reg, Xn, eaReg );
    }

    // long
    if( opmode == 6 )
    {
        if( decImm )sprintf( buf, "eor.l %ld(a%X,d%X),d%X", displ, reg, Xn, eaReg );
        else sprintf( buf, "eor.l $%lX(a%X,d%X),d%X", displ, reg, Xn, eaReg );
    }

    _address += 4;
}

if( mode == 7)
{
    // (xxx).w,dn
    if( eaReg == 0 )
    {
        displ = memory->GetWord( _address + 2 );

        // byte
        if( opmode == 4 )
        {
            sprintf( buf, "eor.b $%lX,d%X", displ, eaReg );
        }
```

```
            // word
            if ( opmode == 5 )
            {
                sprintf ( buf, "eor.w $%lX,d%X", displ, eaReg );
            }

            // long
            if ( opmode == 6 )
            {
                sprintf ( buf, "eor.l $%lX,d%X", displ, eaReg );
            }
            _address += 4;
        }

        // (xxx).l,dn
        if ( eaReg == 1 )
        {
            displ = memory->GetLongword( _address + 2 );

            // byte
            if ( opmode == 4 )
            {
                sprintf ( buf, "eor.b $%lX,d%X", displ, eaReg );
            }

            // word
            if ( opmode == 5 )
            {
                sprintf ( buf, "eor.w $%lX,d%X", displ, eaReg );
            }

            // long
            if ( opmode == 6 )
            {
                sprintf ( buf, "eor.l $%lX,d%X", displ, eaReg );
            }
            _address += 6;
        }
    }

    // OMITTED
    // #data,dn (see ORI)
    // d16(PC),dn
    // d8(PC,xn)

    return _address;
}

unsigned long M68008::tS_eori( unsigned long _address, char *buf )
{
    int size, mode, reg;
    unsigned long displ;
```

```
DATA_REGISTER( imm );

size = GETBITS( _address, 0x00C0, 6 );
mode = GETBITS( _address, 0x0038, 3 );
reg  = GETBITS( _address, 0x0007, 0 );

// Fetch the immediate value
if( size == 0 )
   imm.b = memory->GetByte( _address + 3 );
if( size == 1 )
   imm.w = memory->GetWord( _address + 2 );
if( size == 2 )
   imm.l = memory->GetLongword( _address + 2 );

// #data,dn
if( mode == 0 )
{
   if( size == 0 )
   {
      if( decImm )sprintf( buf, " eori.b #%d,d%X", imm.b, reg );
      else sprintf( buf, " eori.b #$%X,d%X", imm.b, reg );
      _address += 4;
   }
   if( size == 1 )
   {
      if( decImm )sprintf( buf, " eori.w #%d,d%X", imm.w, reg );
      else sprintf( buf, " eori.w #$%X,d%X", imm.w, reg );
      _address += 4;
   }
   if( size == 2 )
   {
      if( decImm )sprintf( buf, " eori.l #%ld,d%X", imm.l, reg );
      else sprintf( buf, " eori.l #$%lX,d%X", imm.l, reg );
      _address += 6;
   }
}

// #data,(an)
if( mode == 2 )
{
   if( size == 0 )
   {
      if( decImm )sprintf( buf, " eori.b #%d,(a%X)", imm.b, reg );
      else sprintf( buf, " eori.b #$%X,(a%X)", imm.b, reg );
      _address += 4;
   }
   if( size == 1 )
   {
      if( decImm )sprintf( buf, " eori.w #%d,(a%X)", imm.w, reg );
      else sprintf( buf, " eori.w #$%X,(a%X)", imm.w, reg );
      _address += 4;
   }
```

```
    if ( size == 2 )
    {
        if ( decImm )sprintf ( buf, " eori.l␣#%ld,(a%X)", imm.l, reg );
        else sprintf ( buf, " eori.l␣#$%lX,(a%X)", imm.l, reg );
        _address += 6;
    }
}

// #data,(an)+
if ( mode == 3 )
{
    if ( size == 0 )
    {
        if ( decImm )sprintf ( buf, " eori.b␣#%d,(a%X)+", imm.b, reg );
        else sprintf ( buf, " eori.b␣#$%X,(a%X)+", imm.b, reg );
        _address += 4;
    }
    if ( size == 1 )
    {
        if ( decImm )sprintf ( buf, " eori.w␣#%d,(a%X)+", imm.w, reg );
        else sprintf ( buf, " eori.w␣#$%X,(a%X)+", imm.w, reg );
        _address += 4;
    }
    if ( size == 2 )
    {
        if ( decImm )sprintf ( buf, " eori.l␣#%ld,(a%X)+", imm.l, reg );
        else sprintf ( buf, " eori.l␣#$%lX,(a%X)+", imm.l, reg );
        _address += 6;
    }
}

// #data,-(an)
if ( mode == 4 )
{
    if ( size == 0 )
    {
        if ( decImm )sprintf ( buf, " eori.b␣#%d,-(a%X)", imm.b, reg );
        else sprintf ( buf, " eori.b␣#$%X,-(a%X)", imm.b, reg );
        _address += 4;
    }
    if ( size == 1 )
    {
        if ( decImm )sprintf ( buf, " eori.w␣#%d,-(a%X)", imm.w, reg );
        else sprintf ( buf, " eori.w␣#$%X,-(a%X)", imm.w, reg );
        _address += 4;
    }
    if ( size == 2 )
    {
        if ( decImm )sprintf ( buf, " eori.l␣#%ld,-(a%X)", imm.l, reg );
        else sprintf ( buf, " eori.l␣#$%lX,-(a%X)", imm.l, reg );
        _address += 6;
    }
```

```
}

// #data,d16(an)
if( mode == 5 )
{
    if( size == 0 )
    {
        displ = (signed short int)memory->GetWord( _address + 4 );
        if( decImm )sprintf( buf, " eori.b #$%X,%ld(a%X)", imm.b, displ, reg );
        else sprintf( buf, " eori.b #$%X,$%lX(a%X)", imm.b, displ, reg );
        _address += 6;
    }
    if( size == 1 )
    {
        displ = (signed short int)memory->GetWord( _address + 4 );
        if( decImm )sprintf( buf, " eori.w #$%X,%ld(a%X)", imm.w, displ, reg );
        else sprintf( buf, " eori.w #$%X,$%lX(a%X)", imm.w, displ, reg );
        _address += 6;
    }
    if( size == 2 )
    {
        displ = (signed short int)memory->GetWord( _address + 6 );
        if( decImm )sprintf( buf, " eori.l #$%lX,%ld(a%X)", imm.l, displ, reg );
        else sprintf( buf, " eori.l #$%lX,$%lX(a%X)", imm.l, displ, reg );
        _address += 8;
    }
}

// #data,d8(an,xn)
if( mode == 6 )
{
    if( size == 0 )
    {
        displ = (signed char)memory->GetByte( _address + 5 )& 0x0fff;
        int Xn = (signed char)memory->GetByte( _address + 5 )>> 12;
        if( decImm )sprintf( buf, " eori.b #%ld,%d(a%X,%d%X)", displ, imm.b, reg, Xn );    else sprintf(
            buf, " eori.b #$%lX,$%X(a%X,%d%X)", displ, imm.b, reg, Xn );
        _address += 6;
    }
    if( size == 1 )
    {
        displ = (signed char)memory->GetByte( _address + 5 )& 0x0fff;
        int Xn = (signed char)memory->GetByte( _address + 5 )>> 12;
        if( decImm )sprintf( buf, " eori.w #%ld,%d(a%X,%d%X)", displ, imm.w, reg, Xn );    else sprintf(
            buf, " eori.w #$%lX,$%X(a%X,%d%X)", displ, imm.w, reg, Xn );
        _address += 6;
    }
    if( size == 2 )
    {
        displ = (signed char)memory->GetByte( _address + 7 )& 0x0fff;
        int Xn = (signed char)memory->GetByte( _address + 7 )>> 12;
        if( decImm )sprintf( buf, " eori.l #$%lX,%ld(a%X,%d%X)", displ, imm.l, reg, Xn );
```

```
        else sprintf( buf, " eori.l #$%lX,$%lX(a%X, d%X)", displ, imm.l, reg, Xn );
        _address += 8;
    }
}

if( mode == 7 )
{
    // #data,(xxx).w
    if( reg == 0 )
    {
        if( size == 0 )
        {
            displ = memory->GetWord( _address + 4 );
            if( decImm )sprintf( buf, " eori.b #%d,$%lX", imm.b, displ );
            else sprintf( buf, " eori.b #$%X,$%lX", imm.b, displ );
            _address += 6;
        }
        if( size == 1 )
        {
            displ = memory->GetWord( _address + 4 );
            if( decImm )sprintf( buf, " eori.w #%d,$%lX", imm.w, displ );
            else sprintf( buf, " eori.w #$%X,$%lX", imm.w, displ );
            _address += 6;
        }
        if( size == 2 )
        {
            displ = memory->GetWord( _address + 6 );
            if( decImm )sprintf( buf, " eori.l #%ld,$%lX", imm.l, displ );
            else sprintf( buf, " eori.l #$%lX,$%lX", imm.l, displ );
            _address += 8;
        }
    }

    // #data,(xxx).l
    if( reg == 1 )
    {
        if( size == 0 )
        {
            displ = memory->GetLongword( _address + 4 );
            if( decImm )sprintf( buf, " eori.b #%d,$%lX", imm.b, displ );
            else sprintf( buf, " eori.b #$%X,$%lX", imm.b, displ );
            _address += 8;
        }
        if( size == 1 )
        {
            displ = memory->GetLongword( _address + 4 );
            if( decImm )sprintf( buf, " eori.w #%d,$%lX", imm.w, displ );
            else sprintf( buf, " eori.w #$%X,$%lX", imm.w, displ );
            _address += 8;
        }
        if( size == 2 )
        {
```

```
            displ = memory->GetLongword( _address + 6 );
            if( decImm )sprintf( buf, " eori.l #%ld,$%lX", imm.l, displ );
            else sprintf( buf, " eori.l #$%lX,$%lX", imm.l, displ );
            _address += 10;
         }
      }
   }

   return _address;
}

unsigned long M68008::tS_eoriCcr( unsigned long _address, char *buf )
{
   int data = GETBITS( _address + 2, 0xffff, 0 );
   if( decImm )sprintf( buf, " eori #%d,Ccr", data );
   else sprintf( buf, " eori #$%X,Ccr", data );

   _address += 4;

   return _address;
}

unsigned long M68008::tS_exg( unsigned long _address, char *buf )
{
   // size is always a longword

   short int opmode, regx, regy;
   regy = GETBITS( _address, 0x7, 0 );
   opmode = GETBITS( _address, 0xF8, 3 );
   regx = GETBITS( _address, 0xE00, 9 );


   // data registers
   if ( opmode == 8 ){
      sprintf( buf, " exg d%X,d%X", regx, regy );
   }

   // address registers
   if ( opmode == 9 ){
      sprintf( buf, " exg a%X,a%X", regx, regy );
   }

   // data and address registers
   // regy is the address register and
   // regx is the data register
   if ( opmode == 17 ){
      sprintf( buf, " exg d%X, a%X", regx, regy );
   }

   _address += 2;
   return _address;
}
```

```
unsigned long M68008::tS_ext( unsigned long _address, char *buf )
{
    short int opmode, reg;
    opmode = GETBITS( _address, 0x1C0, 6 );
    reg = GETBITS( _address, 0x7, 0 );

    if ( opmode == 2 ){
        // byte to word operation
        sprintf ( buf, "ext_d%X", reg );
    }

    if ( opmode == 3 ){
        // word to longword operation
        sprintf ( buf, "ext_d%X", reg );
    }
    _address += 2;

    return _address;
}

unsigned long M68008::tS_illegal( unsigned long _address, char *buf )
{
    sprintf ( buf, " illegal " ) ;

    _address += 2;

    return _address;
}

unsigned long M68008::tS_jmp( unsigned long _address, char *buf ){

    long new_address = _address;
    //displacement = _address + 2 + operand
    //new _address = _address + ( _address + 2 + operand )

    int ch = memory->GetWord( _address );

    // if the mode is 111
    if ( GETBITS( _address, 56, 0 )== 56 ){

        // and the register is 000
        // jmp (xxx).w
        if ( GETBITS( _address, 7, 0 )== 0 ){
            new_address = memory->GetWord( _address + 2 );
            sprintf ( buf, "jmp_$%lX", new_address );
            _address += 4;
            return _address;
        }

        // and the register is 001
        // jmp (www).l
```

```
        if ( GETBITS( _address, 7, 0 )== 1 ){
            new_address = memory->GetLongword( _address + 2 );
            sprintf ( buf, "jmp $%lX", new_address );
            _address += 6;
            return _address;
        }

        // and if the register is 010
        // jmp (d16,PC)
        // OMIT : This instruction is not currently implemented.
        if ( GETBITS( _address, 7, 0 )== 2 ){
            sprintf ( buf, "jmp (d16, PC): Unimplemented" );
            _address += 4;
            return _address;
        }


    }

    // if the mode is 010
    // jmp (An)
    if ( GETBITS( _address, 56, 0 )== 16 ){

        // we need to decode the register number
        int var1;
        var1 = ch & 7;
        sprintf ( buf, "jmp (a%X)", var1 );
        _address += 2;
        return _address;

    }

    // if the mode is 101
    // jmp (d16, An)
    if ( GETBITS( _address, 56, 0 )== 40 ){
        // we need to decode the register number
        int var1;
        var1 = ch & 7;

        int displacement = (signed short int)memory->GetWord( _address + 2 );
        if( decImm )sprintf ( buf, "jmp %d(a%X)", displacement, var1 );
        else sprintf ( buf, "jmp $%X(a%X)", displacement, var1 );
        _address += 4;
        return _address;
    }

    // OMIT: modes 110 and last two 111 instructions are not implemented
    // compiler wouldn't recognise the instructions
    // come back to it later
    return _address;
}
```

```
unsigned long M68008::tS_jsr( unsigned long _address, char *buf ){
   /* The function of the jsr instruction is to preserve the long value of
   the address for the next instruction to the stack and jump to that location.
      PC + instruction length -> Stack
      Stack pointer is updated.l
      Destination address -> PC */

   long new_address = _address;

   // if the mode is 111
   if ( GETBITS( _address, 56, 0 )== 56 ){

      // if the register is 000 then the instruction type is
      // jsr (xxx).w
      if ( GETBITS( _address, 7, 0) == 0 ){

         // before incrementing the _address to the next instruction
         // we must get the destination address
         // long dest = memory->GetWord( _address + 2 );

         // increment the PC to the next instruction.
         _address += 4;

         sprintf ( buf, " jsr $%04lX", _address );
         return _address;

      }

      // if the register is 1 then the instruction type is
      // jsr (xxx).l
      if ( GETBITS( _address, 7, 0) == 1 ){

         // before incrementing the _address to the next instruction
         // we must get the destination address
         // long dest = memory->GetLongword( _address + 2 );

         // increment the PC to the next instruction.
         _address += 6;

         sprintf ( buf, " jsr $%lX", _address );
         return _address;
      }

      // if the register is 010 then the instruction type is
      // jsr (d16, PC)
      if ( GETBITS( _address, 7, 0) == 2 ){
         sprintf ( buf, "Unimplemented");
/*
         printf( "jsr: longword displacement to PC\n" );

         //  before incrementing the _address to the next instruction
         // we get the destination address
```

```
        int displacement = memory->GetWord( _address + 2 );

        if ( displacement >= 32509 && displacement <= 65276) {
           // displacement is negative
           // so we need to calculate the correct displacement in integer
           // so
           displacement -= 65272;
        } else {
           // displacement is positive
           // so we need to calculate the correct displacement in integer
           // so
           displacement -= 65276;
        }

        long dest = displacement + _address;

        // increment the PC to the next instruction
        new_address += 4;

        // Add the long word PC to the stack.
        a[7].l -= 4;                    // first decrement pointer so its pointing at the next space
        memory->SetLongword( _address, a[7].l );    // set longword on stack to be _address

        // destination address -> _address
        new_address = dest;
*/
        new_address += 4;
        return _address;

      }

   }

   // if the mode is 010
   if ( GETBITS( _address, 56, 0 )== 16 ){

      // then the instruction type is
      // jsr (An)
      // we need to decode the register number
      int var1;

      var1 = memory->GetWord( _address )& 7;

      if (var1 >= 8) {
         printf("ERROR!");
      } else {

         sprintf ( buf, " jsr (a%X)", var1 );

         // increment the PC to the next instruction
         _address += 2;
         return _address;
```

```
    }
  }

  // if the mode is 101
  if ( GETBITS( _address, 56, 0 )== 40 ){
    sprintf ( buf, "Unimplemented" );
/*
    a[0].l = 0x106;

    // then the instruction type is
    // jsr (d16,An)
    // we need to decode the register number

    printf( "jmp: word displacement to address register\n" );

    // we need to decode the register number
    int var1;

    var1 = memory->GetWord( _address )& 7;



    if (var1 >= 8) {

      printf( "ERROR!" );

    } else {

      // get the address location of the value pointed to by the adress register
      long dest = a[var1].l;
      dest = memory->GetLongword( dest );
      int displacement = memory->GetWord( _address + 2 );

      if ( displacement >= 32768) {
        // displacement is negative
        // so we need to calculate the correct displacement in integer
        // so
        displacement -= 65536;
      }

      dest += displacement;

      // increment the PC to the next instruction
      new_address += 4;

      // Add the long word PC to the stack.
      a[7].l -= 4;                    // first decrement pointer so its pointing at the next space
      memory->SetLongword( _address, a[7].l );   // set longword on stack to be _address

      // destination address -> _address
      new_address = dest;
```

```
        }
*/
        _address += 4;
    }

    return _address;
}

unsigned long M68008::tS_lea( unsigned long _address, char *buf ){

    int reg, mode, eareg;
    unsigned long disp;
    reg = GETBITS( _address, 0xe00, 9 );
    mode = GETBITS( _address, 0x38, 3 );
    eareg = GETBITS( _address, 0x7, 0 );

    // (An)
    if( mode == 2) {
        sprintf( buf, " lea (a%X),a%X", eareg, reg );
        _address += 2;
    }

    // d16,An
    if ( mode == 5) {
        disp = (signed short int)memory->GetWord( _address + 2 );
        if( decImm )sprintf( buf, " lea %ld(a%X),a%X", disp, eareg, reg );
        else sprintf( buf, " lea $%lX(a%X),a%X", disp, eareg, reg );
        _address += 4;
    }

    // d8(An,Xn)
    if( mode == 6) {
        disp = (signed char)memory->GetByte( _address + 3 );

        if( decImm )sprintf( buf, " lea %ld(a%X,d%X),a%X", disp, eareg, GETBITS( _address + 2, 0xF000, 12 )
            , reg );
        else sprintf( buf, " lea $%lX(a%X,d%X),a%X", disp, eareg, GETBITS( _address + 2, 0xF000, 12 ), reg );
        _address += 4;
    }

    // No thing I can say in here
    if ( mode == 7) {

        // (XXX).W
        if( eareg == 0) {
            sprintf( buf, " lea $%X,a%X", memory->GetWord( _address + 2 ), reg );
            _address += 4;
        }

        // (XXX).L
        if( eareg == 1) {
            sprintf( buf, " lea $%lX,a%X", memory->GetLongword( _address + 2 ), reg );
```

```
            _address += 6;
        }

        /* OMIT:


        // d16(PC)
        if ( eareg == 2) {
            a[ reg ].l = memory->GetLongword( (signed long int)memory->GetWord( _address + 2 )+ _address)
                ;
            _address += 4;
        }

        // d8(PC,Xn)
        if ( eareg == 3) {
        }
        */
    }

    return _address;
}

unsigned long M68008::tS_linkWord( unsigned long _address, char *buf ){
    signed short int displ = memory->GetWord( _address + 2 );
    short int reg = GETBITS( _address, 0x0007, 0 );
    if ( decImm )sprintf ( buf, " link a%X,#%d", reg, displ );
    else sprintf ( buf, " link a%X,#$%X", reg, displ );
    _address += 4;
    return _address;
}

unsigned long M68008::tS_linkLong( unsigned long _address, char *buf ){
    sprintf ( buf, " link : Not Supported" );
    _address += 2;
    return _address;
}

unsigned long M68008::tS_lsl_rReg( unsigned long _address, char *buf ){

    short int reg = GETBITS( pc, 0x0007, 0 );
    short int ir = GETBITS( pc, 0x0020, 5);
    short int size = GETBITS( pc, 0x00c0, 6 );
    short int dr = GETBITS( pc, 0x0100, 8 );
    short int creg = GETBITS( pc, 0x0e00, 9 );
    // long int contents = 0;

    short int count;

    if ( ir == 0 ){

        // gets the count
        if ( creg == 0 ){
```

```
        count = 8;
    } else {
        count = creg;
    }

} else {
    creg = d[ creg ].l;
}

// if the operation is a byte.
if( size == 0 ){
    // shifting right.
    if( dr == 0 ){
        if( ir == 0 ){
            if( decImm )sprintf( buf, " lsr.b #%d,d%X", creg, reg );
            else sprintf( buf, " lsr.b #$%X,d%X", creg, reg );
        } else {
            sprintf( buf, " lsr.b d%X,d%X", creg, reg );
        }
    } else {
        // shifting left
        if( ir == 0 ){
            if( decImm )sprintf( buf, " lsl.b #%d,d%X", creg, reg );
            else sprintf( buf, " lsl.b #$%X,d%X", creg, reg );
        } else {
            sprintf( buf, " lsl.b d%X,d%X", creg, reg );
        }
    }
}

if( size == 1 ){
    // shifting right.
    if( dr == 0 ){
        if( ir == 0 ){
            if( decImm )sprintf( buf, " lsr.w #%d,d%X", creg, reg );
            else sprintf( buf, " lsr.w #$%X,d%X", creg, reg );
        } else {
            sprintf( buf, " lsr.w d%X,d%X", creg, reg );
        }
    } else {
        // shifting left
        if( ir == 0 ){
            if( decImm )sprintf( buf, " lsl.w #%d,d%X", creg, reg );
            else sprintf( buf, " lsl.w #$%X,d%X", creg, reg );
        } else {
            sprintf( buf, " lsl.w d%X,d%X", creg, reg );
        }
    }
}

if( size == 2 ){
    // shifting right.
```

```
        if( dr == 0 ){
            if( ir == 0 ){
                if( decImm )sprintf( buf, " lsr.l #%d,d%X", creg, reg );
                else sprintf( buf, " lsr.l #$%X,d%X", creg, reg );
            } else {
                sprintf( buf, " lsr.l d%X,d%X", creg, reg );
            }
        } else {
            // shifting left
            if( ir == 0 ){
                if( decImm )sprintf( buf, " lsl.l #%d,d%X", creg, reg );
                else sprintf( buf, " lsl.l #$%X,d%X", creg, reg );
            } else {
                sprintf( buf, " lsl.l d%X,d%X", creg, reg );
            }
        }
    }

    pc += 2;
    return _address;
}

unsigned long M68008::tS_lsl_rMem( unsigned long _address, char *buf ){
    // This method specifically only deals with word ops.

    short int dr = GETBITS( pc, 0x0100, 8 );
    short int reg = GETBITS( pc, 0x0007, 0 );
    short int mode = GETBITS( pc, 0x0038, 3 );
    // short int contents = 0;

    // if <ea> is (an).
    if( mode == 2 ){

        // shifting right.
        if( dr == 0 ){
            sprintf( buf, " lsr.w (a%X)", reg );
        } else {
            // shifting left
            sprintf( buf, " lsl.w (a%X)", reg );
        }
        pc += 2;
    }

    // if <ea> is (an)+
    if( mode == 3 ){

        // shifting right.
        if( dr == 0 ){
            sprintf( buf, " lsr.w (a%X)+", reg );
        } else {
            // shifting left
            sprintf( buf, " lsl.w (a%X)+", reg );
```

```
    }

    pc += 2;
}

// if <ea> is -(an)
if( mode == 4 ){

    // shifting right.
    if( dr == 0 ){
        sprintf( buf, " lsr.w -(a%X)", reg );
    } else {
        // shifting left
        sprintf( buf, " lsl.w -(a%X)", reg );
    }

    pc += 2;
}

// if <ea> is d16(an)
if( mode == 5 ){
    int displ = (signed short int)memory->GetWord( pc + 2 );

    // shifting right.
    if( dr == 0 ){
        if( decImm )sprintf( buf, " lsr.w %d(a%X)", displ, reg );
        else sprintf( buf, " lsr.w $%X(a%X)", displ, reg );
    } else {
        // shifting left
        if( decImm )sprintf( buf, " lsl.w %d(a%X)", displ, reg );
        else sprintf( buf, " lsl.w $%X(a%X)", displ, reg );
    }

    pc += 4;
}

// if <ea> is d8(an,Xn)
if( mode == 6 ){
    int displ = (signed char)memory->GetByte( pc + 3 );

    // shifting right.
    if( dr == 0 ){
        if( decImm )sprintf( buf, " lsr.w %d(a%X,d%X)", displ, reg, GETBITS( pc + 2, 0xf000, 12) );
        else sprintf( buf, " lsr.w $%X(a%X,d%X)", displ, reg, GETBITS( pc + 2, 0xf000, 12) );
    } else {
        // shifting left
        if( decImm )sprintf( buf, " lsl.w %d(a%X,d%X)", displ, reg, GETBITS( pc + 2, 0xf000, 12));
        else sprintf( buf, " lsl.w $%X(a%X,d%X)", displ, reg, GETBITS( pc + 2, 0xf000, 12));
    }

    pc += 4;
}
```

```
    // if <ea> is a real address.
    if( mode == 7 ){
        long int address = 0;
        if( reg == 0 ){
            // word address
            address = memory->GetWord( pc + 2 );

            // shifting right.
            if( dr == 0 ){
                sprintf( buf, " lsr.w $%04lX", address );
            } else {
                // shifting left
                sprintf( buf, " lsl.w $%04lX", address );
            }

            pc += 4;

        }

        if( reg == 1 ){
            // long address
            address = memory->GetLongword( pc + 2 );

            // shifting right.
            if( dr == 0 ){
                sprintf( buf, " lsr.w $%08lX", address );
            } else {
                // shifting left
                sprintf( buf, " lsl.w $%08lX", address );
            }

            pc += 6;

        }
    }
    return _address;
}

unsigned long M68008::tS_move( unsigned long _address, char *buf )
{
    char buffer[80];

    short int dreg, dmode, sreg, smode, size;
    unsigned long displ;
    sreg = GETBITS( _address, 0x7, 0 );
    smode = GETBITS( _address, 0x38, 3 );
    dmode = GETBITS( _address, 0x1c0, 6 );
    dreg = GETBITS( _address, 0xe00, 9 );
    size = GETBITS( _address, 0x3000, 12 );

    // if the source is a data register.
```

```
if ( smode == 0 ){

    // a byte operation
    if(size == 1) {

        sprintf ( buffer , " move.b␣d%X%c", sreg, 0 );
    }

    // a word operation
    if ( size == 3 ){
        sprintf ( buffer , " move.w␣d%X%c", sreg, 0 );
    }

    // a long operation
    if ( size == 2 ){
        sprintf ( buffer , " move.l␣d%X%c", sreg, 0 );
    }
    _address += 2;
}

// if the source is an address register
if ( smode == 1 ){

    // a byte operation
    if ( size == 1 ){
        sprintf ( buffer , " move.b␣a%X%c", sreg, 0 );
    }

    // a word operation
    if ( size == 3 ){
        sprintf ( buffer , " move.w␣a%X%c", sreg, 0 );
    }

    // a long operation
    if ( size == 2 ){
        sprintf ( buffer , " move.l␣a%X%c", sreg, 0 );
    }
    _address += 2;
}

// if the source is an address pointer
if ( smode == 2 ){

    // a byte operation
    if ( size == 1 ){
        sprintf ( buffer , " move.b␣(a%X)%c", sreg, 0 );
    }

    // a word operation
    if ( size == 3 ){
        sprintf ( buffer , " move.w␣(a%X)%c", sreg, 0 );
    }
```

```
    // a long operation
    if ( size == 2 ){
        sprintf ( buffer , " move.l␣(a%X)%c", sreg, 0 );
    }
    ␣address += 2;
}

// if the source is an ( address pointer )+
if ( smode == 3 ){

    // a byte operation
    if ( size == 1 ){
        sprintf ( buffer , " move.b␣(a%X)+%c", sreg, 0 );
    }

    // a word operation
    if ( size == 3 ){
        sprintf ( buffer , " move.w␣(a%X)+%c", sreg, 0 );
    }

    // a long operation
    if ( size == 2 ){
        sprintf ( buffer , " move.l␣(a%X)+%c", sreg, 0 );
    }
    ␣address += 2;
}

// if the source is an −( address pointer )
if ( smode == 4 ){

    // a byte operation
    if ( size == 1 ){
        sprintf ( buffer , " move.b␣−(a%X)%c", sreg, 0 );
    }

    // a word operation
    if ( size == 3 ){
        sprintf ( buffer , " move.w␣−(a%X)%c", sreg, 0 );
        }

    // a long operation
    if ( size == 2 ){
        sprintf ( buffer , " move.l␣−(a%X)%c", sreg, 0 );
    }
    ␣address += 2;
}


// d16(An)
// if the source is a displacement to address register
if ( smode == 5 ){
```

```
    int disp = memory->GetWord( _address + 2 );

    // a byte operation
    if ( size == 1 ){
        if( decImm )sprintf( buffer, " move.b %d(a%X)%c", disp, sreg, 0 );
        else sprintf( buffer, " move.b $%X(a%X)%c", disp, sreg, 0 );
    }

    // a word operation
    if ( size == 3 ){
        if( decImm )sprintf( buffer, " move.w %d(a%X)%c", disp, sreg, 0 );
        else sprintf( buffer, " move.w $%X(a%X)%c", disp, sreg, 0 );
        }

    // a long operation
    if ( size == 2 ){
        if( decImm )sprintf( buffer, " move.l %d(a%X)%c", disp, sreg, 0 );
        else sprintf( buffer, " move.l $%X(a%X)%c", disp, sreg, 0 );
    }
    _address += 4;
}

// d8(An, Xn)
// if the source is a displacement to address register , and data register
if ( smode == 6 ){
    int disp = memory->GetWord( _address + 2 )& 0x0FFF;
    int Xn = memory->GetWord( _address + 2 )>> 12;


    // a byte operation
    if ( size == 1 ){

        if( decImm )sprintf( buffer, " move.b %d(a%X, d%X)%c", disp, sreg, Xn, 0 );
        else sprintf( buffer, " move.b $%X(a%X, d%X)%c", disp, sreg, Xn, 0 );
    }

    // a word operation
    if ( size == 3 ){
        if( decImm )sprintf( buffer, " move.w %d(a%X, d%X)%c", disp, sreg, Xn, 0 );
        else sprintf( buffer, " move.w $%X(a%X, d%X)%c", disp, sreg, Xn, 0 );
    }

    // a long operation
    if ( size == 2 ){
        if( decImm )sprintf( buffer, " move.l %d(a%X, d%X)%c", disp, sreg, Xn, 0 );
        else sprintf( buffer, " move.l $%X(a%X, d%X)%c", disp, sreg, Xn, 0 );
    }
    _address += 4;
}

// mode is 7
```

```
if ( smode == 7 ){

   // (XXX).W
   if ( sreg == 0 ){

      displ = memory->GetWord( _address + 2 );

      // a byte operation
      if ( size == 1 ){
         sprintf ( buffer , " move.b $%lX%c", displ, 0 );
      }

      // a word operation
      if ( size == 3 ){
         sprintf ( buffer , " move.w $%lX%c", displ, 0 );
      }

      // a long operation
      if ( size == 2 ){
         sprintf ( buffer , " move.l $%lX%c", displ, 0 );
      }
      _address += 4;
   }

   // (XXX),L
   if ( sreg == 1 ){

      displ = memory->GetLongword( _address + 2 );

      // a byte operation
      if ( size == 1 ){
         sprintf ( buffer , " move.b $%lX%c", displ, 0 );
      }

      // a word operation
      if ( size == 3 ){
         sprintf ( buffer , " move.w $%lX%c", displ, 0 );
      }

      // a long operation
      if ( size == 2 ){
         sprintf ( buffer , " move.l $%lX%c", displ, 0 );
      }
      _address += 6;
   }

   if ( sreg == 4 ){
      // int disp = memory->GetWord( _address + 2 );

      // a byte operation
      if ( size == 1) {
         int disp = memory->GetWord( _address + 2 );
```

```
            if( decImm )sprintf( buffer, "move.b #%d%c", disp, 0 );
            else sprintf( buffer, "move.b #$%X%c", disp, 0 );
            _address += 4;
        }

        if ( size == 3 ){
            int disp = memory->GetWord( _address + 2 );

            if( decImm )sprintf( buffer, "move.w #%d%c", disp, 0 );
            else sprintf( buffer, "move.w #$%X%c", disp, 0 );
            _address += 4;
        }

        if ( size == 2) {
            int disp = memory->GetLongword( _address + 2 );

            if( decImm )sprintf( buffer, "move.l #%d%c", disp, 0 );
            else sprintf( buffer, "move.l #$%X%c", disp, 0 );
            _address += 6;
        }
    }
}


/* OMIT--------
   d16(PC) and d8(PC, Xn)
   */



//-------------------------------------------------------------------//


// destination is a data register
if( dmode == 0 ){

    // a byte operation
    if(size == 1) {
        sprintf( buf, "%s,d%X", buffer, dreg );
    }

    // a word operation
    if ( size == 3 ){
        sprintf( buf, "%s,d%X", buffer, dreg );
    }

    // a long operation
    if ( size == 2 ){
        sprintf( buf, "%s,d%X", buffer, dreg );
    }
}
```

```
// destination is an address register pointer
if ( dmode == 2 ){

   // a byte operation
   if(size == 1) {
      sprintf ( buf, "%s,(a%X)", buffer, dreg );
   }

   // a word operation
   if ( size == 3 ){
      sprintf ( buf, "%s,(a%X)", buffer, dreg );
   }

   // a long operation
   if ( size == 2 ){
      sprintf ( buf, "%s,(a%X)", buffer, dreg );
   }
}

// destination is an (address pointer)+
if ( dmode == 3 ){

   // a byte operation
   if(size == 1) {
      sprintf ( buf, "%s,(a%X)+", buffer, dreg );
   }

   // a word operation
   if ( size == 3 ){
      sprintf ( buf, "%s,(a%X)+", buffer, dreg );
   }

   // a long operation
   if ( size == 2 ){
      sprintf ( buf, "%s,(a%X)+", buffer, dreg );
   }
}

// destination is an -(address pointer)
if ( dmode == 4 ){

   // a byte operation
   if(size == 1) {
      sprintf ( buf, "%s,-(a%X)", buffer, dreg );
   }

   // a word operation
   if ( size == 3 ){
      sprintf ( buf, "%s,-(a%X)", buffer, dreg );
   }
```

```
    // a long operation
    if ( size == 2 ){
        sprintf ( buf, "%s,-(a%X)", buffer, dreg );
    }
}

// destination is displacement to address pointer
if ( dmode == 5 ){

    displ = memory->GetWord( _address );

    // a byte operation
    if(size == 1) {
        if( decImm )sprintf( buf, "%s,%ld(a%X)", buffer, displ, dreg );
        else sprintf( buf, "%s,$%lX(a%X)", buffer, displ, dreg );
    }

    // a word operation
    if ( size == 3 ){
        if( decImm )sprintf( buf, "%s,%ld(a%X)", buffer, displ, dreg );
        else sprintf( buf, "%s,$%lX(a%X)", buffer, displ, dreg );
    }

    // a long operation
    if ( size == 2 ){
        if( decImm )sprintf( buf, "%s,%ld(a%X)", buffer, displ, dreg );
        else sprintf( buf, "%s,$%lX(a%X)", buffer, displ, dreg );
    }
    _address += 2;
}


// destination is displacement to address pointer and data registers
if ( dmode == 6 ){

    int disp = memory->GetWord( _address )& 0x0FFF;
    int Xn = memory->GetWord( _address )>> 12;

    if( decImm )sprintf( buf, "%s,%d(a%X,d%X)", buffer, disp, Xn, dreg );
    else sprintf( buf, "%s,$%X(a%X,d%X)", buffer, disp, Xn, dreg );

    _address += 2;
}


if ( dmode == 7 ){

    // destination is (XXX).W
    if ( dreg == 0 ){

        displ = memory->GetWord( _address );
```

```
            sprintf ( buf, "%s,$%lX", buffer, displ ) ;

            _address += 2;
        }

        // destination is (XXX).L
        if ( dreg == 1 ){

            displ = memory->GetLongword( _address );


            sprintf ( buf, "%s,$%lX", buffer, displ ) ;

            _address += 4;
        }
    }

    return _address;
}

unsigned long M68008::tS_movea( unsigned long _address, char *buf )
{
    char buffer[80];
    short int dreg, sreg, smode, size;
    unsigned long displ;

    sreg = GETBITS( _address, 0x7, 0 );
    smode = GETBITS( _address, 0x38, 3 );
    dreg = GETBITS( _address, 0xe00, 9 );
    size = GETBITS( _address, 0x3000, 12 );

    //source
    if( smode == 0 ){

        // a word operation
        if ( size == 3 ){
            sprintf ( buffer, "movea.w d%X%c", sreg, 0 );
        }

        // a long operation
        if ( size == 2 ){
            sprintf ( buffer, "movea.l d%X%c", sreg, 0 );
        }
        _address += 2;
    }

    // if the source is an address register
    if ( smode == 1 ){

        // a word operation
        if ( size == 3 ){
```

```
        sprintf ( buffer , " movea.w␣a%X%c", sreg, 0 );
    }

    // a long operation
    if ( size == 2 ){
        sprintf ( buffer , " movea.l␣a%X%c", sreg, 0 );
    }
    ␣address += 2;
}

// if the source is an address pointer
if ( smode == 2 ){

    // a word operation
    if ( size == 3 ){
        sprintf ( buffer , " movea.w␣(a%X)%c", sreg, 0 );
    }

    // a long operation
    if ( size == 2 ){
        sprintf ( buffer , " movea.l␣(a%X)%c", sreg, 0 );
    }
    ␣address += 2;
}

// if the source is an ( address pointer )+
if ( smode == 3 ){

    // a word operation
    if ( size == 3 ){
        sprintf ( buffer , " movea.w␣(a%X)+%c", sreg, 0 );
    }

    // a long operation
    if ( size == 2 ){
        sprintf ( buffer , " movea.l␣(a%X)+%c", sreg, 0 );
    }
    ␣address += 2;
}

// if the source is an −( address pointer )
if ( smode == 4 ){

    // a word operation
    if ( size == 3 ){
        sprintf ( buffer , " movea.w␣−(a%X)%c", sreg, 0 );
    }

    // a long operation
    if ( size == 2 ){
        sprintf ( buffer , " movea.l␣−(a%X)%c", sreg, 0 );
    }
```

```
    _address += 2;
}


// d16(An)
// if the source is a displacement to address register
if ( smode == 5 ){

    displ = (signed short int)memory->GetWord( _address + 2 );

    // a word operation
    if ( size == 3 ){
        if( decImm )sprintf( buffer, "movea.w %ld(a%X)%c", displ, sreg, 0 );
        else sprintf( buffer, "movea.w $%lX(a%X)%c", displ, sreg, 0 );
    }

    // a long operation
    if ( size == 2 ){
        if( decImm )sprintf( buffer, "movea.l %ld(a%X)%c", displ, sreg, 0 );
        else sprintf( buffer, "movea.l $%lX(a%X)%c", displ, sreg, 0 );
    }
    _address += 4;
}

// d8(An, Xn)
// if the source is a displacement to address register , and data register
if ( smode == 6 ){

    displ = (signed char)memory->GetByte( _address + 3 );
    int var = GETBITS( _address + 2, 0xF000, 12 );
    // a word operation
    if ( size == 3 ){
        if( decImm )sprintf( buffer, "movea.w %ld(a%X, d%X)%c", displ, sreg, var, 0 );
        else sprintf( buffer, "movea.w $%lX(a%X, d%X)%c", displ, sreg, var, 0 );
    }

    // a long operation
    if ( size == 2 ){
        if( decImm )sprintf( buffer, "movea.l %ld(a%X, d%X)%c", displ, sreg, var, 0 );
        else sprintf( buffer, "movea.l $%lX(a%X, d%X)%c", displ, sreg, var, 0 );
    }
    _address += 4;
}

// mode is 7
if ( smode == 7 ){

    // (XXX).W
    if ( sreg == 0 ){

        displ = memory->GetWord( _address + 2 );
```

```
        // a word operation
        if ( size == 3 ){
            sprintf ( buffer , " movea.w $%lX%c", displ, 0 );
        }

        // a long operation
        if ( size == 2 ){
            sprintf ( buffer , " movea.l $%lX%c", displ, 0 );
        }
        address += 4;
    }

    // (XXX),L
    if ( sreg == 1 ){

        displ = memory->GetLongword( address + 2 );

        // a word operation
        if ( size == 3 ){
            sprintf ( buffer , " movea.w $%lX%c", displ, 0 );
        }

        // a long operation
        if ( size == 2 ){
            sprintf ( buffer , " movea.l $%lX%c", displ, 0 );
        }
        address += 6;
    }

    // #<data>
    if ( sreg == 4 ){
        // a word operation
        if ( size == 3 ){
            if( decImm )sprintf ( buffer , " movea.w #%d%c", memory->GetWord( address + 2 ), 0 );
            else sprintf ( buffer , " movea.w #$%X%c", memory->GetWord( address + 2 ), 0 );
            address += 4;
        }

        // a long operation
        if ( size == 2 ){
            if( decImm )sprintf ( buffer , " movea.l #%ld%c", memory->GetLongword( address + 2 ), 0 );
            else sprintf ( buffer , " movea.l #$%lX%c", memory->GetLongword( address + 2 ), 0 );
            address += 6;
        }
    }
}


/* OMIT-------
   d16(PC) and d8(PC, Xn)
*/
```

```
    // destination
    // destination is an address register
    // a word operation
    if ( size == 3 ){
        sprintf ( buf, "%s,a%X", buffer, dreg );
    }

    // a long operation
    if ( size == 2 ){
        sprintf ( buf, "%s,a%X", buffer, dreg );
    }


    return _address;
}

unsigned long M68008::tS_moveFromCcr( unsigned long _address, char *buf )
{
    short int reg, mode;
    unsigned long displ;
    reg = GETBITS( _address, 0x7, 0 );
    mode = GETBITS( _address, 0x38, 3 );


    // -> dn
    if ( mode == 0 ){
        sprintf ( buf, "move_CCR,d%X", reg );
        _address += 2;
    }

    // -> (An)
    if ( mode == 2 ){
        sprintf ( buf, "move_CCR,a%X", reg );
        _address += 2;
    }

    // -> (An)+
    if ( mode == 3 ){
        sprintf ( buf, "move_CCR,(a%X)+", reg );
        _address += 2;
    }

    // -> -(An)
    if ( mode == 4 ){
        sprintf ( buf, "move_CCR,-(a%X)", reg );
        _address += 2;
    }

    // -> (d16, An)
    if ( mode == 5 ){
        displ = (signed short int)memory->GetWord( _address + 2 );
        if( decImm )sprintf ( buf, "move_CCR,%ld(a%X)", displ, reg );
```

```
        else sprintf ( buf, " move_CCR,$%lX(a%X)", displ, reg );
        _address += 4;
    }

    // -> d8(An, Xn)
    if ( mode == 6 ){
        displ = (signed char)memory->GetByte( _address + 3 );
        int var = GETBITS( _address + 2, 0xF000, 12 );
        if ( decImm )sprintf ( buf, " move_CCR,%ld(a%X,_d%X)", displ, reg, var );
        else sprintf ( buf, " move_CCR,$%lX(a%X,_d%X)", displ, reg, var );
        _address += 4;
    }

    // -> (XXX).W
    if ( mode == 7 && reg == 0 ){
        displ = memory->GetWord( _address + 2 );
        sprintf ( buf, " move_CCR,$%lX", displ );
        _address += 4;
    }

    // -> (XXX).L
    if ( mode == 7 && reg == 1 ){
        displ = memory->GetLongword( _address + 2 );
        sprintf ( buf, " move_CCR,$%lX", displ );
        _address += 6;
    }

    return _address;
}

unsigned long M68008::tS_moveToCcr( unsigned long _address, char *buf )
{
    // always a word operation

    short int operand;
    short int reg, mode;
    unsigned long displ;
    reg = GETBITS( _address, 0x7, 0 );
    mode = GETBITS( _address, 0x38, 3 );

    // (Dn) -> CCR
    if ( mode == 0 ){
        sprintf ( buf, " move_d%X,CCR", reg );
        _address += 2;
    }


    // (An) -> CCR
    if ( mode == 2 ){
        sprintf ( buf, " move_(a%X),CCR", reg );
        _address += 2;
    }
```

```
// (An)+ -> CCR
if ( mode == 3 ){
    sprintf ( buf, "move (a%X)+,CCR", reg );
    _address += 2;
}

// -(An) -> CCR
if ( mode == 4 ){
    sprintf ( buf, "move -(a%X),CCR", reg );
    _address += 2;
}

// (d16, An) -> CCR
if ( mode == 5 ){
    displ = (signed short int)memory->GetWord( _address + 2 );
    if ( decImm )sprintf ( buf, "move %ld(a%X),CCR", displ, reg );
    else sprintf ( buf, "move $%lX(a%X),CCR", displ, reg );
    _address += 4;
}

// d8(An, Xn) -> CCR
if ( mode == 6 ){
    displ = (signed char)memory->GetByte( _address + 3 );
    int var = GETBITS( _address + 2, 0xF000, 12 );
    if ( decImm )sprintf ( buf, "move %ld(a%X, d%X),CCR", displ, reg, var );
    else sprintf ( buf, "move $%lX(a%X, d%X),CCR", displ, reg, var );
    _address += 4;
}

// (XXX).W -> CCR
if ( mode == 7 && reg == 0 ){
    displ = memory->GetWord( _address + 2 );
    sprintf ( buf, "move $%lX,CCR", displ );
    _address += 4;
}

// (XXX).L -> CCR
if ( mode == 7 && reg == 1 ){
    displ = memory->GetLongword( _address + 2 );
    sprintf ( buf, "move $%lX,CCR", displ );
    _address += 6;
}

// #<data> -> CCR
if ( mode == 7 && reg == 4 ){
    operand = memory->GetWord( _address + 2 );
    if ( decImm )sprintf ( buf, "move #%d,CCR", operand );
    else sprintf ( buf, "move #$%X,CCR", operand );
    _address += 4;
}
```

```
    /* OMIT: (d16, PC), d8(PC, Xn) */

    return _address;
}

unsigned long M68008::tS_movem( unsigned long _address, char *buf )
{
    short int dr, size , mode, reg;
    dr = GETBITS( _address, 0x400, 10 );
    size = GETBITS( _address, 0x40, 6 );
    mode = GETBITS( _address, 0x38, 3 );
    reg = GETBITS( _address, 0x7, 0 );
    short int mask = memory->GetWord( _address + 2 );
    short int count = 0;


    // to calculate the number of registers marked
    for ( int i = 0; i < 16; i++ ){
        int temp = mask >> i;
        temp = temp & 0x0001;

        if( temp == 1 ){
            count++;
        }
    }




    _address += 2;
    return _address;
}

unsigned long M68008::tS_movep( unsigned long _address, char *buf )
{
    sprintf( buf, "movep: Unimplemented" );
    _address += 2;
    return _address;
}

unsigned long M68008::tS_moveq( unsigned long _address, char *buf )
{
    char data = GETBITS( _address, 0xFF, 0 );
    short int reg = GETBITS( _address, 0xE00, 9 );

    sprintf( buf, "moveq #%i,d%X", data, reg );

    _address += 2;

    return _address;
}
```

```
unsigned long M68008::tS_mulsWord( unsigned long _address, char *buf )
{
    unsigned long displ;
    short int sreg, smode, dreg;
    sreg = GETBITS( _address, 0x7, 0 );
    smode = GETBITS( _address, 0x38, 3 );
    // destination is always a register
    dreg = GETBITS( _address, 0xE00, 9 );

    // dn x dn -> dn
    if ( smode == 0 ){
        sprintf( buf, "muls.w␣d%X,d%X", sreg, dreg );
        _address += 2;
    }

    // dn x (An) -> dn
    if ( smode == 2 ){
        sprintf( buf, "muls.w␣(a%X),d%X", sreg, dreg );
        _address += 2;
    }

    // dn x (An)+ -> dn
    if ( smode == 3 ){
        sprintf( buf, "muls.w␣(a%X)+,d%X", sreg, dreg );
        _address += 2;
    }

    // dn x -(An) -> dn
    if (smode == 4 ){
        sprintf( buf, "muls.w␣-(a%X),d%X", sreg, dreg );
        _address += 2;

    }

    // dn x d16(An) -> dn
    if (smode == 5 ){
        if( decImm )sprintf( buf, "muls.w␣%d(a%X),d%X", (signed short int)memory->GetWord( _address
            + 2 ), sreg, dreg );
        else sprintf( buf, "muls.w␣$%X(a%X),d%X", (signed short int)memory->GetWord( _address + 2 ),
            sreg, dreg );
        _address += 4;
    }

    // dn x d8(An, Xn) -> dn
    if ( smode == 6 ){
        if( decImm )sprintf( buf, "muls.w␣%d(a%X,d%X),d%X", (signed char)memory->GetByte( _address
            + 3 ), sreg, GETBITS( _address + 2, 0xF000, 12 ), dreg );
        else sprintf( buf, "muls.w␣$%X(a%X,d%X),d%X", (signed char)memory->GetByte( _address + 3 ),
            sreg, GETBITS( _address + 2, 0xF000, 12 ), dreg );
        _address += 4;
    }
```

```
    // dn x (xxx).w -> dn
    if ( smode == 7 && sreg == 0 ){
        displ = memory->GetWord( _address + 2 );
        sprintf ( buf, " muls.w $%lX,d%X", displ, dreg );
        _address += 4;
    }

    // dn x (xxx).L -> dn
    if (smode == 7 && sreg == 1 ){
        displ = memory->GetLongword( _address + 2 );
        sprintf ( buf, " muls.w $%lX,d%X", displ, dreg );
        _address += 6;
    }

    // dn x #<xxx> -> dn
    if ( smode == 7 && sreg == 4 ){
        displ = memory->GetWord( _address + 2 );
        if ( decImm )sprintf ( buf, " muls.w #%ld,d%X", displ, dreg );
        else sprintf ( buf, " muls.w #$%lX,d%X", displ, dreg );
        _address += 4;
    }


    /*
    OMIT: d16(PC) and d8(An, Xn)
    */

    return _address;
}

unsigned long M68008::tS_mulsLong( unsigned long _address, char *buf )
{
    sprintf ( buf, " mulsLong: Unimplemented\n" );
    // Invalid Size

    return _address;
}

unsigned long M68008::tS_muluWord( unsigned long _address, char *buf )
{
    unsigned long displ;
    short int sreg, smode, dreg;
    sreg = GETBITS( _address, 0x7, 0 );
    smode = GETBITS( _address, 0x38, 3 );
    // destination is always a register
    dreg = GETBITS( _address, 0xE00, 9 );

    // dn x dn -> dn
    if ( smode == 0 ){
        sprintf ( buf, " mulu.w d%X,d%X", sreg, dreg );
        _address += 2;
    }
```

```
// dn x (An) -> dn
if ( smode == 2 ){
    sprintf ( buf, "mulu.w (a%X),d%X", sreg, dreg );
    address += 2;
}

// dn x (An)+ -> dn
if ( smode == 3 ){
    sprintf ( buf, "mulu.w (a%X)+,d%X", sreg, dreg );
    address += 2;
}

// dn x -(An) -> dn
if (smode == 4 ){
    sprintf ( buf, "mulu.w -(a%X),d%X", sreg, dreg );
    address += 2;
}

// dn x d16(An) -> dn
if (smode == 5 ){
    displ = (signed short int)memory->GetWord( address + 2 );
    if ( decImm )sprintf ( buf, "mulu.w %ld(a%X),d%X", displ, sreg, dreg );
    else sprintf ( buf, "mulu.w $%lX(a%X),d%X", displ, sreg, dreg );
    address += 4;
}

// dn x d8(An, Xn) -> dn
if ( smode == 6 ){
    displ = (signed char)memory->GetByte( address + 3 );
    int var = GETBITS( address + 2, 0xF000, 12 );
    if ( decImm )sprintf ( buf, "mulu.w %ld(a%X, d%X),d%X", displ, sreg, var, dreg );
    else sprintf ( buf, "mulu.w $%lX(a%X, d%X),d%X", displ, sreg, var, dreg );
    address += 4;
}

// dn x (xxx).w -> dn
if ( smode == 7 && sreg == 0 ){
    displ = memory->GetWord( address + 2 );
    sprintf ( buf, "mulu.w $%lX,d%X", displ, dreg );
    address += 4;
}

// dn x (xxx).L -> dn
if (smode == 7 && sreg == 1 ){
    displ = memory->GetLongword( address + 2 );
    sprintf ( buf, "mulu.w $%lX,d%X", displ, dreg );
    address += 6;
}

// dn x #<xxx> -> dn
if ( smode == 7 && sreg == 4 ){
```

```
        displ = memory->GetWord( _address + 2 );
        if ( decImm )sprintf ( buf, "mulu.w #%ld,d%X", displ, dreg );
        else sprintf ( buf, "mulu.w #$%lX,d%X", displ, dreg );
        _address += 4;
    }

    return _address;
}

unsigned long M68008::tS_muluLong( unsigned long _address, char *buf )
{
    sprintf ( buf, "muluLong: Invalid Size" );
    // Invalid Size

    return _address;
}

unsigned long M68008::tS_nbcd( unsigned long _address, char *buf )
{
    sprintf ( buf, "nbcd: Unimplemented" );

    return _address;
}

unsigned long M68008::tS_neg( unsigned long _address, char *buf )
{
    // stores the _addressode in ch
    int ch = memory->GetWord( _address );
    short int dreg;
    dreg = GETBITS( _address, 0x7, 0 );

    //
    //
    // If mode is 000 then it is a data register that is being accessed
    if ( ( ch & 0x38) == 0) {

        // if the size is 0 then it is a byte operation
        if ( ( ch & 0xc0) == 0) {
            sprintf ( buf, "neg.b d%X", dreg );
            _address += 2;
        }
        // if the size is 1 then it is a word operation
        if ( ( ( ch & 0xc0) >> 6 )== 1) {
            sprintf ( buf, "neg.w d%X", dreg );
            _address += 2;
        }
        // if the size is 2 then it is a long operation
        if ( ( ( ch & 0xc0) >> 6 )== 2) {
            sprintf ( buf, "neg.l d%X", dreg );
            _address += 2;
        }
    }
```

```
//
//
//
// If mode is 010 then it is a address register that is being accessed
if ( ( ( ch & 0x38) >> 3 )== 2) {

   // if the size is 0 then it is a byte operation
   if ( ( ch & 0xc0) == 0) {
      sprintf ( buf, " neg.b (a%X)", dreg );
      address += 2;
   }

   // if the size is 1 then it is a word operation
   if ( ( (  ch & 0xc0 ) >> 6 )== 1) {
      sprintf ( buf, " neg.w (a%X)", dreg );
      address += 2;
   }

   // if the size is 2 then it is a long operation
   if ( ( (  ch & 0xc0 ) >> 6 )== 2) {
      sprintf ( buf, " neg.l (a%X)", dreg );
      address += 2;
   }
}

//
//
//
// If mode is 011 then it is a (address register )+ that is being accessed
if ( ( ( ch & 0x38) >> 3 )== 3) {

   // if the size is 0 then it is a byte operation
   if ( ( (  ch & 0xc0 ) >> 6 )== 0) {
      sprintf ( buf, " neg.b (a%X)+", dreg );
      address += 2;
   }

   // if the size is 1 then it is a word operation
   if ( ( (  ch & 0xc0 ) >> 6 )== 1) {
      sprintf ( buf, " neg.w (a%X)+", dreg );
      address += 2;
   }

   // if the size is 2 then it is a long operation
   if ( ( (  ch & 0xc0 ) >> 6 )== 2) {
      sprintf ( buf, " neg.l (a%X)+", dreg );
      address += 2;
   }
}

//
```

```
//
//
// If mode is 100 then it is a -(address register) that is being accessed
if ( ( ( ch & 0x38) >> 3 )== 4) {

   // if the size is 0 then it is a byte operation
   if ( ( (  ch & 0xc0 ) >> 6 )== 0) {
      sprintf ( buf, "neg.b -(a%X)", dreg );
      _address += 2;
   }

   // if the size is 1 then it is a word operation
   if ( ( (  ch & 0xc0 ) >> 6 )== 1) {
      sprintf ( buf, "neg.w -(a%X)", dreg );
      _address += 2;
   }

   // if the size is 2 then it is a long operation
   if ( ( (  ch & 0xc0 ) >> 6 )== 2) {
      sprintf ( buf, "neg.l -(a%X)", dreg );
      _address += 2;
   }
}

//
//
//
// If mode is 111 then it is a (XXX) instruction
if ( ( ( ch & 0x38) >> 3 )== 7 ){
   // if register is 0 then it is in the form (XXX).W
   if ( (  ch & 0x7 ) == 0 ){

      // if the size is 0 then it is a byte operation
      if ( ( (  ch & 0xc0 ) >> 6 )== 0) {

         short int displ = memory->GetWord( _address + 2 );
         sprintf ( buf, "neg.b $%X", displ );
      }

      // if the size is 1 then it is a word operation
      if ( ( (  ch & 0xc0 ) >> 6 )== 1) {

         short int displ = memory->GetWord( _address + 2 );
         sprintf ( buf, "neg.w $%X", displ );
      }

      // if the size is 2 then it is a long operation
      if ( ( (  ch & 0xc0 ) >> 6 )== 2) {

         int displ = memory->GetWord( _address + 2 );
         sprintf ( buf, "neg.l $%X", displ );
      }
```

```
        _address += 4;
    }

    /*

        OMIT
        THIS INSTRUCTION HAS NOT BEEN TESTED
        BECAUSE WE ARE UNABLE TO GET ROBODEV TO COMPILE IT

    */

    // if register is 1 then it is in the form (XXX).L
    if ( ( ch & 0x7 ) == 1 ){
        sprintf ( buf, " neg (XXX).l: Unimplemented" );
        /*
        // if the size is 0 then it is a byte operation
        if ( ( ( ch & 0xc0 ) >> 6 )== 0) {

            long int displ = memory->GetLongword( _address + 2 );
            sprintf ( buf, "neg.b $%X", displ );
        }

        // if the size is 1 then it is a word operation
        if ( ( ( ch & 0xc0 ) >> 6 )== 1) {

            long int displ = memory->GetLongword( _address + 2 );
            sprintf ( buf, "neg.w $%X", displ );
        }

        // if the size is 2 then it is a long operation
        if ( ( ( ch & 0xc0 ) >> 6 )== 2) {

            long int displ = memory->GetLongword( _address + 2 );
            sprintf ( buf, "neg.l $%X", displ );
        }
    _address += 6;
    */
    _address += 2;
    }
  }

  return _address;
}

unsigned long M68008::tS_negx( unsigned long _address, char *buf )
{
  // stores the _addressode in ch
  int ch = memory->GetWord( _address );
  short int dreg;
  dreg = GETBITS( _address, 0x7, 0 );
  //
  //
```

```
// If mode is 000 then it is a data register that is being accessed
if ( ( ch & 0x38) == 0) {

    // if the size is 0 then it is a byte operation
    if ( ( ch & 0xc0) == 0) {

        sprintf ( buf, "negx.b d%X", dreg );
        _address += 2;
    }
    // if the size is 1 then it is a word operation
    if ( ( ( ch & 0xc0) >> 6 )== 1) {

        sprintf ( buf, "negx.w d%X", dreg );
        _address += 2;
    }
    // if the size is 2 then it is a long operation
    if ( ( ( ch & 0xc0) >> 6 )== 2) {
        sprintf ( buf, "negx.l d%X", dreg );
        _address += 2;
    }
}

//
//
//
// If mode is 010 then it is a address register that is being accessed
if ( ( ( ch & 0x38) >> 3 )== 2) {

    // if the size is 0 then it is a byte operation
    if ( ( ch & 0xc0) == 0) {
        sprintf ( buf, "negx.b (a%X)", dreg );
        _address += 2;
    }

    // if the size is 1 then it is a word operation
    if ( ( ( ch & 0xc0 ) >> 6 )== 1) {
        sprintf ( buf, "negx.w (a%X)", dreg );
        _address += 2;
    }

    // if the size is 2 then it is a long operation
    if ( ( ( ch & 0xc0 ) >> 6 )== 2) {
        sprintf ( buf, "negx.l (a%X)", dreg );
        _address += 2;
    }
}

//
//
//
// If mode is 011 then it is a (address register )+ that is being accessed
if ( ( ( ch & 0x38) >> 3 )== 3) {
```

```
    // if the size is 0 then it is a byte operation
    if ( ( (  ch & 0xc0 ) >> 6 )== 0) {
        sprintf ( buf, "negx.b (a%X)+", dreg );
        _address += 2;
    }

    // if the size is 1 then it is a word operation
    if ( ( (  ch & 0xc0 ) >> 6 )== 1) {
        sprintf ( buf, "negx.w (a%X)+", dreg );
        _address += 2;
    }

    // if the size is 2 then it is a long operation
    if ( ( (  ch & 0xc0 ) >> 6 )== 2) {
        sprintf ( buf, "negx.l (a%X)+", dreg );
        _address += 2;
    }
}

//
//
//
// If mode is 100 then it is a -(address register) that is being accessed
if ( ( ( ch & 0x38) >> 3 )== 4) {

    // if the size is 0 then it is a byte operation
    if ( ( (  ch & 0xc0 ) >> 6 )== 0) {
        sprintf ( buf, "negx.b -(a%X)", dreg );
        _address += 2;
        return _address;
    }

    // if the size is 1 then it is a word operation
    if ( ( (  ch & 0xc0 ) >> 6 )== 1) {
        sprintf ( buf, "negx.w -(a%X)", dreg );
        _address += 2;
        return _address;
    }

    // if the size is 2 then it is a long operation
    if ( ( (  ch & 0xc0 ) >> 6 )== 2) {
        sprintf ( buf, "negx.l -(a%X)", dreg );
        _address += 2;
        return _address;
    }
}

//
//
//
// If mode is 111 then it is a (XXX) instruction
```

```
    if ( ( ( ch & 0x38) >> 3 )== 7 ){
       // if register is 0 then it is in the form (XXX).W
       if ( ( ch & 0x7 ) == 0 ){

          // if the size is 0 then it is a byte operation
          if ( ( ( ch & 0xc0 ) >> 6 )== 0) {

             short int displ = memory->GetWord( _address + 2 );
             sprintf ( buf, "negx.b $%X", displ );
             _address += 4;
             return _address;
          }

          // if the size is 1 then it is a word operation
          if ( ( ( ch & 0xc0 ) >> 6 )== 1) {

             short int displ = memory->GetWord( _address + 2 );
             sprintf ( buf, "negx.w $%X", displ );
                _address += 4;
             return _address;

          }

          // if the size is 2 then it is a long operation
          if ( ( ( ch & 0xc0 ) >> 6 )== 2) {

             int displ = memory->GetWord( _address + 2 );
             sprintf ( buf, "negx.l $%X", displ );
                _address += 4;
             return _address;

          }

       }

    }

    return _address;
}

unsigned long M68008::tS_nop( unsigned long _address, char *buf )
{
    sprintf ( buf, "nop" );
    _address += 2;

    return _address;
}

/* OMITTED is the displacement to the address register operand to this instruction. */
unsigned long M68008::tS_not( unsigned long _address, char *buf ){
    long int new_address = _address;
```

```
/*
short int instruction = memory->GetWord( _address );
short int size = GETBITS( _address, 192, 6);
short int dmode = GETBITS( _address, 56, 3);
short int dreg = GETBITS( _address, 7, 0);
*/

// if the mode is a word address
if ( ( GETBITS( _address, 56, 0) == 56 )){

   // if the register is 000, it is a word address.
   if (( GETBITS( _address, 7, 0) == 0 )){
      printf("not with word operand.\n");

      if (GETBITS( _address, 192, 6) == 0) {
         // byte operation
         // get memory location
         long int operand = memory->GetWord( _address + 2);
         sprintf ( buf, "not.b $%8lX", operand );

      }

      if (GETBITS( _address, 192, 6) == 1) {
         // word operation
         // get memory location
         long int operand = memory->GetWord( _address + 2);
         sprintf ( buf, "not.w $%lX", operand );
      }

      if (GETBITS( _address, 192, 6) == 2) {
         // long operation
         // get memory location
         long int operand = memory->GetWord( _address + 2);
         sprintf ( buf, "not.l $%lX", operand );
      }

      _address += 4;
   }

   //
   //
   // if the register is 001, it is a long address.
   if (( GETBITS( _address, 7, 0) == 1 )){
      printf("not with long operand.\n");

      if (GETBITS( _address, 192, 6) == 0) {
         // byte operation
         // get memory location
         long int operand = memory->GetLongword( _address + 2);
         sprintf ( buf, "not.b $%lX", operand );
      }
```

```
        if (GETBITS( _address, 192, 6) == 1) {
            // word operation
            // get memory location
            long int operand = memory->GetLongword( _address + 2);
            sprintf ( buf, "not.w $%lX", operand );
        }

        if (GETBITS( _address, 192, 6) == 2) {
            // long operation
            // get memory location
            long int operand = memory->GetLongword( _address + 2);
            sprintf ( buf, "not.l $%lX", operand );
        }
        _address += 6;
    }

}

//
//
// if the mode is a data register
if ( (GETBITS( _address, 56, 0) == 0 )){
    // decode the register number
    int var = GETBITS( _address, 7, 0);

    if (GETBITS( _address, 192, 6) == 0) {
        // byte op
        printf("This is a byte, not, data register op\n");
        // get the contents of the data register
        sprintf ( buf, "not.b d%X", var );
    }

    if (GETBITS( _address, 192, 6) == 1) {
        // word op
        printf("This is a word, not, data register op\n");
        sprintf ( buf, "not.w d%X", var );
    }


    if (GETBITS( _address, 192, 6) == 2) {
        // long op
        printf("This is a long, not, data register op\n");
        // get the contents of the data register
        sprintf ( buf, "not.l d%X", var );
    }
    new_address += 2;
}

//
//
// if the mode is an address register pointer
if ( (GETBITS( _address, 56, 3) == 2 )){
```

```
    // decode the register number
    int var = GETBITS( _address, 7, 0);

    if (GETBITS( _address, 192, 6) == 0) {
        sprintf ( buf, "not.b␣(a%X)", var );
    }


    if (GETBITS( _address, 192, 6) == 1) {
        sprintf ( buf, "not.w␣(a%X)", var );
    }


    if (GETBITS( _address, 192, 6) == 2) {
        sprintf ( buf, "not.l␣(a%X)", var );
    }

    new_address += 2;
}

//
//
// if the mode is an post-increment address register pointer
if ( (GETBITS( _address, 56, 3) == 3 )){

    // decode the register number
    int var = GETBITS( _address, 7, 0);
    if (GETBITS( _address, 192, 6) == 0) {
        sprintf ( buf, "not.b␣(a%X)+", var );
    }


    if (GETBITS( _address, 192, 6) == 1) {
        sprintf ( buf, "not.w␣(a%X)+", var );
    }


    if (GETBITS( _address, 192, 6) == 2) {
        sprintf ( buf, "not.l␣(a%X)+", var );
    }
    new_address += 2;
}

    // if the mode is an pre-decrement address register pointer
if ( (GETBITS( _address, 56, 3) == 4 )){

    // decode the register number
    int var = GETBITS( _address, 7, 0);
    if (GETBITS( _address, 192, 6) == 0) {
        sprintf ( buf, "not.b␣-(a%X)", var );
    }
```

```
        if (GETBITS( _address, 192, 6) == 1) {
            sprintf ( buf, "not.w (a%X)", var );
        }


        if (GETBITS( _address, 192, 6) == 2) {
            sprintf ( buf, "not.l (a%X)", var );
        }
        new_address += 2;
    }
    _address = new_address;

    return _address;
}

/* OMIT
    (d16, An) and (d16, An, Xn) modes have been omitted awaiting further information
*/
```

1.8 MInstrO-Z.cpp

The execution methods of the instructions starting with $O \cdots Z$.

```
/*
    Motorola 68008 Simulator
    (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
    Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

    This class represents the Motorola 68008 Microprocessor
    Instruction definitions O-Z
*/

#include <iostream.h>
#include <math.h>
#include <stdio.h>
#include "M68008.h"

#define MSB8 0x80
#define MSB16 0x8000
#define MSB32 0x80000000

#define GETBITS( addr, mask, shift )( ( memory->GetWord( addr )& mask )>> shift )
#define SETNZB( src ){ ccr.n = ( (signed char)src < 0 ); ccr.z = ( src == 0 ); }
#define SETNZW( src ){ ccr.n = ( (signed short)src < 0 ); ccr.z = ( src == 0 ); }
#define SETNZL( src ){ ccr.n = ( (signed long)src < 0 ); ccr.z = ( src == 0 ); }
#define IPOW( n, m )( (unsigned long int)pow( n, m ))
#define SUB_SETV( S, D, R, M )ccr.v = ( ( ~( S & M )& ( D & M )& ~( R & M )| ( S & M )& ~( D & M )
    & ( R & M ))& M )? 1 : 0;
#define SUB_SETC( S, D, R, M )ccr.c = ccr.x = ( ( ( S & M )& ~( D & M )| ( R & M )& ~( D & M )| ( S &
    M )& ( R & M ))& M ) ? 1 : 0;
```

```
void M68008::or()
{
    int reg, opmode, mode, eaReg;
    DATA_REGISTER( result );
    unsigned long displ;


    reg    = GETBITS( pc, 0x0E00, 9 );
    opmode = GETBITS( pc, 0x01C0, 6 );
    mode   = GETBITS( pc, 0x0038, 3 );
    eaReg  = GETBITS( pc, 0x0007, 0 );

    // dn,Dn
    if( mode == 0 )
    {
        // byte
        if( opmode == 0 )
        {
            result.b = d[ eaReg ].b | d[ reg ].b;
            SETNZB( result.b );
            ccr.v = ccr.c = 0;
            d[ reg ].b = result.b;
        }

        // word
        if( opmode == 1 )
        {
            result.w = d[ eaReg ].w | d[ reg ].w;
            SETNZW( result.w );
            ccr.v = ccr.c = 0;
            d[ reg ].w = result.w;
        }

        // long
        if( opmode == 2 )
        {
            result.l = d[ eaReg ].l | d[ reg ].l;
            SETNZL( result.l );
            ccr.v = ccr.c = 0;
            d[ reg ].l = result.l;
        }

        pc += 2;
    }

    // (an),dn
    if( mode == 2 )
    {
        // byte
        if( opmode == 0 )
        {
```

```
    result.b = memory->GetByte( a[ eaReg ].l )| d[ reg ].b;
    SETNZB( result.b );
    ccr.v = ccr.c = 0;
    d[ reg ].b = result.b;
}

// word
if( opmode == 1 )
{
    result.w = memory->GetWord( a[ eaReg ].l )| d[ reg ].w;
    SETNZW( result.w );
    ccr.v = ccr.c = 0;
    d[ reg ].w = result.w;
}

// long
if( opmode == 2 )
{
    result.l = memory->GetLongword( a[ eaReg ].l )| d[ reg ].l;
    SETNZL( result.l );
    ccr.v = ccr.c = 0;
    d[ reg ].l = result.l;
}

// byte
if( opmode == 4 )
{
    result.b = memory->GetByte( a[ eaReg ].l )| d[ reg ].b;
    SETNZB( result.b );
    ccr.v = ccr.c = 0;
    memory->SetByte( result.b, a[ eaReg ].l );
}

// word
if( opmode == 5 )
{
    result.w = memory->GetWord( a[ eaReg ].l )| d[ reg ].w;
    SETNZW( result.w );
    ccr.v = ccr.c = 0;
    memory->SetWord( result.w, a[ eaReg ].l );
}

// long
if( opmode == 6 )
{
    result.l = memory->GetLongword( a[ eaReg ].l )| d[ reg ].l;
    SETNZL( result.l );
    ccr.v = ccr.c = 0;
    memory->SetLongword( result.l, a[ eaReg ].l );
}

pc += 2;
```

```
}

// (an)+,dn
if( mode == 3 )
{
    // byte
    if( opmode == 0 )
    {
        result.b = memory->GetByte( a[ eaReg ].l )| d[ reg ].b;
        SETNZB( result.b );
        ccr.v = ccr.c = 0;
        d[ reg ].b = result.b;
        a[ eaReg ].l += 1;
    }

    // word
    if( opmode == 1 )
    {
        result.w = memory->GetWord( a[ eaReg ].l )| d[ reg ].w;
        SETNZW( result.w );
        ccr.v = ccr.c = 0;
        d[ reg ].w = result.w;
        a[ eaReg ].l += 2;
    }

    // long
    if( opmode == 2 )
    {
        result.l = memory->GetLongword( a[ eaReg ].l )| d[ reg ].l;
        SETNZL( result.l );
        ccr.v = ccr.c = 0;
        d[ reg ].l = result.l;
        a[ eaReg ].l += 4;
    }

    // byte
    if( opmode == 4 )
    {
        result.b = memory->GetByte( a[ eaReg ].l )| d[ reg ].b;
        SETNZB( result.b );
        ccr.v = ccr.c = 0;
        memory->SetByte( result.b, a[ eaReg ].l );
        a[ eaReg ].l += 1;
    }

    // word
    if( opmode == 5 )
    {
        result.w = memory->GetWord( a[ eaReg ].l )| d[ reg ].w;
        SETNZW( result.w );
        ccr.v = ccr.c = 0;
        memory->SetWord( result.w, a[ eaReg ].l );
```

```
        a[ eaReg ].l += 2;
    }

    // long
    if( opmode == 6 )
    {
        result.l = memory->GetLongword( a[ eaReg ].l )| d[ reg ].l;
        SETNZL( result.l );
        ccr.v = ccr.c = 0;
        memory->SetLongword( result.l, a[ eaReg ].l );
        a[ eaReg ].l += 4;
    }

    pc += 2;
}

// -(an),dn
if( mode== 4 )
{
    // byte
    if( opmode == 0 )
    {
        a[ eaReg ].l -= 1;
        result.b = memory->GetByte( a[ eaReg ].l )| d[ reg ].b;
        SETNZB( result.b );
        ccr.v = ccr.c = 0;
        d[ reg ].b = result.b;
    }

    // word
    if( opmode == 1 )
    {
        a[ eaReg ].l -= 2;
        result.w = memory->GetWord( a[ eaReg ].l )| d[ reg ].w;
        SETNZW( result.w );
        ccr.v = ccr.c = 0;
        d[ reg ].w = result.w;
    }

    // long
    if( opmode == 2 )
    {
        a[ eaReg ].l -= 4;
        result.l = memory->GetLongword( a[ eaReg ].l )| d[ reg ].l;
        SETNZL( result.l );
        ccr.v = ccr.c = 0;
        d[ reg ].l = result.l;
    }

    // byte
    if( opmode == 4 )
    {
```

```
        a[ eaReg ].l -= 1;
        result.b = memory->GetByte( a[ eaReg ].l )| d[ reg ].b;
        SETNZB( result.b );
        ccr.v = ccr.c = 0;
        memory->SetByte( result.b, a[ eaReg ].l );
    }

    // word
    if( opmode == 5 )
    {
        a[ eaReg ].l -= 2;
        result.w = memory->GetWord( a[ eaReg ].l )| d[ reg ].w;
        SETNZW( result.w );
        ccr.v = ccr.c = 0;
        memory->SetWord( result.w, a[ eaReg ].l );
    }

    // long
    if( opmode == 6 )
    {
        a[ eaReg ].l -= 4;
        result.l = memory->GetLongword( a[ eaReg ].l )| d[ reg ].l;
        SETNZL( result.l );
        ccr.v = ccr.c = 0;
        memory->SetLongword( result.l, a[ eaReg ].l );
    }

    pc += 2;
}

// d16(an),dn
if( mode == 5 )
{
    displ = (signed short int)memory->GetWord( pc + 2 )+ a[ eaReg ].l;

    // byte
    if( opmode == 0 )
    {
        result.b = memory->GetByte( displ )| d[ reg ].b;
        SETNZB( result.b );
        ccr.v = ccr.c = 0;
        d[ reg ].b = result.b;
    }

    // word
    if( opmode == 1 )
    {
        result.w = memory->GetWord( displ )| d[ reg ].w;
        SETNZW( result.w );
        ccr.v = ccr.c = 0;
        d[ reg ].w = result.w;
    }
```

```
    // long
    if( opmode == 2 )
    {
        result.l = memory->GetLongword( displ )| d[ reg ].l;
        SETNZL( result.l );
        ccr.v = ccr.c = 0;
        d[ reg ].l = result.l;
    }

    // byte
    if( opmode == 4 )
    {
        result.b = memory->GetByte( displ )| d[ reg ].b;
        SETNZB( result.b );
        ccr.v = ccr.c = 0;
        memory->SetByte( result.b, displ );
    }

    // word
    if( opmode == 5 )
    {
        result.w = memory->GetWord( displ )| d[ reg ].w;
        SETNZW( result.w );
        ccr.v = ccr.c = 0;
        memory->SetWord( result.w, displ );
    }

    // long
    if( opmode == 6 )
    {
        result.l = memory->GetLongword( displ )| d[ reg ].l;
        SETNZL( result.l );
        ccr.v = ccr.c = 0;
        memory->SetLongword( result.l, displ );
    }

    pc += 4;
}

// d8(an,xn),dn
if( mode == 6 )
{
    displ = (signed char)memory->GetByte( pc + 3 )+ a[ eaReg ].l
        + d[ GETBITS( pc + 2, 0xF000, 12 )].l;

    // byte
    if( opmode == 0 )
    {
        result.b = memory->GetByte( displ )| d[ reg ].b;
        SETNZB( result.b );
        ccr.v = ccr.c = 0;
```

```
        d[ reg ].b = result.b;
    }

    // word
    if( opmode == 1 )
    {
        result.w = memory->GetWord( displ )| d[ reg ].w;
        SETNZW( result.w );
        ccr.v = ccr.c = 0;
        d[ reg ].w = result.w;
    }

    // long
    if( opmode == 2 )
    {
        result.l = memory->GetLongword( displ )| d[ reg ].l;
        SETNZL( result.l );
        ccr.v = ccr.c = 0;
        d[ reg ].l = result.l;
    }

    // byte
    if( opmode == 4 )
    {
        result.b = memory->GetByte( displ )| d[ reg ].b;
        SETNZB( result.b );
        ccr.v = ccr.c = 0;
        memory->SetByte( result.b, displ );
    }

    // word
    if( opmode == 5 )
    {
        result.w = memory->GetWord( displ )| d[ reg ].w;
        SETNZW( result.w );
        ccr.v = ccr.c = 0;
        memory->SetWord( result.w, displ );
    }

    // long
    if( opmode == 6 )
    {
        result.l = memory->GetLongword( displ )| d[ reg ].l;
        SETNZL( result.l );
        ccr.v = ccr.c = 0;
        memory->SetLongword( result.l, displ );
    }

    pc += 4;
}

if( mode == 7)
```

```
{
    // (xxx).w,dn
    if ( eaReg == 0 )
    {
        displ = memory->GetWord( pc + 2 );

        // byte
        if ( opmode == 0 )
        {
            result.b = memory->GetByte( displ )| d[ reg ].b;
            SETNZB( result.b );
            ccr.v = ccr.c = 0;
            d[ reg ].b = result.b;
        }

        // word
        if ( opmode == 1 )
        {
            result.w = memory->GetWord( displ )| d[ reg ].w;
            SETNZW( result.w );
            ccr.v = ccr.c = 0;
            d[ reg ].w = result.w;
        }

        // long
        if ( opmode == 2 )
        {
            result.l = memory->GetLongword( displ )| d[ reg ].l;
            SETNZL( result.l );
            ccr.v = ccr.c = 0;
            d[ reg ].l = result.l;
        }

        // byte
        if ( opmode == 4 )
        {
            result.b = memory->GetByte( displ )| d[ reg ].b;
            SETNZB( result.b );
            ccr.v = ccr.c = 0;
            memory->SetByte( result.b, displ );
        }

        // word
        if ( opmode == 5 )
        {
            result.w = memory->GetWord( displ )| d[ reg ].w;
            SETNZW( result.w );
            ccr.v = ccr.c = 0;
            memory->SetWord( result.w, displ );
        }

        // long
```

```
    if( opmode == 6 )
    {
        result.l = memory->GetLongword( displ )| d[ reg ].l;
        SETNZL( result.l );
        ccr.v = ccr.c = 0;
        memory->SetLongword( result.l, displ );
    }
    pc += 4;
}

// (xxx).l,dn
if( eaReg == 1 )
{
    displ = memory->GetLongword( pc + 2 );

    // byte
    if( opmode == 0 )
    {
        result.b = memory->GetByte( displ )| d[ reg ].b;
        SETNZB( result.b );
        ccr.v = ccr.c = 0;
        d[ reg ].b = result.b;
    }

    // word
    if( opmode == 1 )
    {
        result.w = memory->GetWord( displ )| d[ reg ].w;
        SETNZW( result.w );
        ccr.v = ccr.c = 0;
        d[ reg ].w = result.w;
    }

    // long
    if( opmode == 2 )
    {
        result.l = memory->GetLongword( displ )| d[ reg ].l;
        SETNZL( result.l );
        ccr.v = ccr.c = 0;
        d[ reg ].l = result.l;
    }

    // byte
    if( opmode == 4 )
    {
        result.b = memory->GetByte( displ )| d[ reg ].b;
        SETNZB( result.b );
        ccr.v = ccr.c = 0;
        memory->SetByte( result.b, displ );
    }

    // word
```

```
            if ( opmode == 5 )
            {
                result.w = memory->GetWord( displ )| d[ reg ].w;
                SETNZW( result.w );
                ccr.v = ccr.c = 0;
                memory->SetWord( result.w, displ );
            }

            // long
            if ( opmode == 6 )
            {
                result.l = memory->GetLongword( displ )| d[ reg ].l;
                SETNZL( result.l );
                ccr.v = ccr.c = 0;
                memory->SetLongword( result.l, displ );
            }
            pc += 6;
        }
    }

    // OMITTED
    // #data,dn (see ORI)
    // d16(PC),dn
    // d8(PC,xn)
}

void M68008::ori()
{
    long int size, mode, reg;
    unsigned long displ;
    DATA_REGISTER( imm );
    DATA_REGISTER( result );


    size = GETBITS( pc, 0x00C0, 6 );
    mode = GETBITS( pc, 0x0038, 3 );
    reg  = GETBITS( pc, 0x0007, 0 );

    // Fetch the immediate value
    if ( size == 0 )
        imm.b = memory->GetByte( pc + 3 );
    if ( size == 1 )
        imm.w = memory->GetWord( pc + 2 );
    if ( size == 2 )
        imm.l = memory->GetLongword( pc + 2 );

    // #data,dn
    if ( mode == 0 )
    {
        if ( size == 0 )
        {
            result.b = imm.b | d[ reg ].b;
```

```
        SETNZB( result.b );
        ccr.c = ccr.v = 0;
        d[ reg ].b = result.b;
        pc += 4;
    }
    if( size == 1 )
    {
        result.w = imm.w | d[ reg ].w;
        SETNZW( result.w );
        ccr.c = ccr.v = 0;
        d[ reg ].w = result.w;
        pc += 4;
    }
    if( size == 2 )
    {
        result.l = imm.l | d[ reg ].l;
        SETNZL( result.l );
        ccr.c = ccr.v = 0;
        d[ reg ].l = result.l;
        pc += 6;
    }
}

// #data,(an)
if( mode == 2 )
{
    if( size == 0 )
    {
        result.b = imm.b | memory->GetByte( a[ reg ].l );
        SETNZB( result.b );
        ccr.c = ccr.v = 0;
        memory->SetByte( result.b, a[ reg ].l );
        pc += 4;
    }
    if( size == 1 )
    {
        result.w = imm.w | memory->GetWord( a[ reg ].l );
        SETNZW( result.w );
        ccr.c = ccr.v = 0;
        memory->SetWord( result.w, a[ reg ].l );
        pc += 4;
    }
    if( size == 2 )
    {
        result.l = imm.l | memory->GetLongword( a[ reg ].l );
        SETNZL( result.l );
        ccr.c = ccr.v = 0;
        memory->SetLongword( result.l, a[ reg ].l );
        pc += 6;
    }
}
```

```
// #data,(an)+
if ( mode == 3 )
{
    if ( size == 0 )
    {
        result.b = imm.b | memory->GetByte( a[ reg ].l );
        SETNZB( result.b );
        ccr.c = ccr.v = 0;
        memory->SetByte( result.b, a[ reg ].l );
        pc += 4;
        a[ reg ].l += 1;
    }
    if ( size == 1 )
    {
        result.w = imm.w | memory->GetWord( a[ reg ].l );
        SETNZW( result.w );
        ccr.c = ccr.v = 0;
        memory->SetWord( result.w, a[ reg ].l );
        pc += 4;
        a[ reg ].l += 2;
    }
    if ( size == 2 )
    {
        result.l = imm.l | memory->GetLongword( a[ reg ].l );
        SETNZL( result.l );
        ccr.c = ccr.v = 0;
        memory->SetLongword( result.l, a[ reg ].l );
        pc += 6;
        a[ reg ].l += 4;
    }
}

// #data,-(an)
if ( mode == 4 )
{
    if ( size == 0 )
    {
        a[ reg ].l -= 1;
        result.b = imm.b | memory->GetByte( a[ reg ].l );
        SETNZB( result.b );
        ccr.c = ccr.v = 0;
        memory->SetByte( result.b, a[ reg ].l );
        pc += 4;
    }
    if ( size == 1 )
    {
        a[ reg ].l -= 2;
        result.w = imm.w | memory->GetWord( a[ reg ].l );
        SETNZW( result.w );
        ccr.c = ccr.v = 0;
        memory->SetWord( result.w, a[ reg ].l );
        pc += 4;
```

```
    }
    if( size == 2 )
    {
        a[ reg ].l -= 4;
        result.l = imm.l | memory->GetLongword( a[ reg ].l );
        SETNZL( result.l );
        ccr.c = ccr.v = 0;
        memory->SetLongword( result.l, a[ reg ].l );
        pc += 6;
    }
}

// #data,d16(an)
if( mode == 5 )
{
    if( size == 0 )
    {
        displ = (signed short int)memory->GetWord( pc + 4 )+ a[ reg ].l;
        result.b = imm.b | memory->GetByte( displ );
        SETNZB( result.b );
        ccr.c = ccr.v = 0;
        memory->SetByte( result.b, displ );
        pc += 6;
    }
    if( size == 1 )
    {
        displ = (signed short int)memory->GetWord( pc + 4 )+ a[ reg ].l;
        result.w = imm.w | memory->GetWord( displ );
        SETNZW( result.w );
        ccr.c = ccr.v = 0;
        memory->SetWord( result.w, displ );
        pc += 6;
    }
    if( size == 2 )
    {
        displ = (signed short int)memory->GetWord( pc + 6 )+ a[ reg ].l;
        result.l = imm.l | memory->GetLongword( displ );
        SETNZL( result.l );
        ccr.c = ccr.v = 0;
        memory->SetLongword( result.l, displ );
        pc += 8;
    }
}

// #data,d8(an,xn)
if( mode == 6 )
{
    if( size == 0 )
    {
        displ = (signed char)memory->GetByte( pc + 5 )+ a[ reg ].l
            + d[ GETBITS( pc + 4, 0xF000, 12 )].l;
        result.b = imm.b | memory->GetByte( displ );
```

```
            SETNZB( result.b );
            ccr.c = ccr.v = 0;
            memory->SetByte( result.b, displ );
            pc += 6;
        }
        if( size == 1 )
        {
            displ = (signed char)memory->GetByte( pc + 5 )+ a[ reg ].l
                + d[ GETBITS( pc + 4, 0xF000, 12 )].l;
            result.w = imm.w | memory->GetWord( displ );
            SETNZW( result.w );
            ccr.c = ccr.v = 0;
            memory->SetWord( result.w, displ );
            pc += 6;
        }
        if( size == 2 )
        {
            displ = (signed char)memory->GetByte( pc + 7 )+ a[ reg ].l
                + d[ GETBITS( pc + 6, 0xF000, 12 )].l;
            result.l = imm.l | memory->GetLongword( displ );
            SETNZL( result.l );
            ccr.c = ccr.v = 0;
            memory->SetLongword( result.l, displ );
            pc += 8;
        }
    }

    if( mode == 7 )
    {
        // #data,(xxx).w
        if( reg == 0 )
        {
            if( size == 0 )
            {
                displ = memory->GetWord( pc + 4 );
                result.b = imm.b | memory->GetByte( displ );
                SETNZB( result.b );
                ccr.c = ccr.v = 0;
                memory->SetByte( result.b, displ );
                pc += 6;
            }
            if( size == 1 )
            {
                displ = memory->GetWord( pc + 4 );
                result.w = imm.w | memory->GetWord( displ );
                SETNZW( result.w );
                ccr.c = ccr.v = 0;
                memory->SetWord( result.w, displ );
                pc += 6;
            }
            if( size == 2 )
            {
```

```
            displ = memory->GetWord( pc + 6 );
            result.l = imm.l | memory->GetLongword( displ );
            SETNZL( result.l );
            ccr.c = ccr.v = 0;
            memory->SetLongword( result.l, displ );
            pc += 8;
        }
    }

    // #data,(xxx).l
    if( reg == 1 )
    {
        if( size == 0 )
        {
            displ = memory->GetLongword( pc + 4 );
            result.b = imm.b | memory->GetByte( displ );
            SETNZB( result.b );
            ccr.c = ccr.v = 0;
            memory->SetByte( result.b, displ );
            pc += 8;
        }
        if( size == 1 )
        {
            displ = memory->GetLongword( pc + 4 );
            result.w = imm.w | memory->GetWord( displ );
            SETNZW( result.w );
            ccr.c = ccr.v = 0;
            memory->SetWord( result.w, displ );
            pc += 8;
        }
        if( size == 2 )
        {
            displ = memory->GetLongword( pc + 6 );
            result.l = imm.l | memory->GetLongword( displ );
            SETNZL( result.l );
            ccr.c = ccr.v = 0;
            memory->SetLongword( result.l, displ );
            pc += 10;
        }
    }
  }
}

void M68008::oriCcr()
{

  ccr.x |= GETBITS( pc + 2, 0x0010, 4 );
  ccr.n |= GETBITS( pc + 2, 0x0008, 3 );
  ccr.z |= GETBITS( pc + 2, 0x0004, 2 );
  ccr.v |= GETBITS( pc + 2, 0x0002, 1 );
  ccr.c |= GETBITS( pc + 2, 0x0001, 0 );
```

```
    pc += 4;
}

void M68008::pea()
{
    int mode, reg;
    char *buf;

    mode = GETBITS( pc, 0x0038, 3 );
    reg  = GETBITS( pc, 0x0007, 0 );

    a [ 7 ]. l -= 4;


    // (an)
    if ( mode == 2 )
    {
        memory->SetLongword( a[ reg ].l, a[ 7 ].l );
        pc += 2;

        buf = new char[ 10 ];
        sprintf ( buf, "(a%d)", reg );
        PUSH( buf, 4 );
    }

    // d16(an)
    if ( mode == 5 )
    {
        memory->SetLongword( a[ reg ].l +
            (signed short int)memory->GetWord( pc + 2 ), a[ 7 ].l );
        pc += 4;

        buf = new char[ 32 ];
        sprintf ( buf, "$%X(a%d)", memory->GetWord( pc + 2 ), reg );
        PUSH( buf, 4 );
    }

    if ( mode == 6 )
    // d8(an,xn)
    {
        memory->SetLongword( a[ reg ].l +
            (signed char)memory->GetByte( pc + 3 )+ d[ GETBITS( pc + 2, 0xF000, 12 )].l,
            a [ 7 ]. l );
        pc += 4;

        buf = new char[ 32 ];
        sprintf ( buf, "$%X(a%d,d%d)", memory->GetByte( pc + 3 ), reg, GETBITS( pc + 2, 0xF000, 12 ));
        PUSH( buf, 4 );
    }

    // (xxx).w
    if ( ( mode == 7 )&& ( reg == 0 ))
```

```
    {
        memory->SetLongword( memory->GetWord( pc + 2 ), a[ 7 ].l );
        pc += 4;

        buf = new char[ 32 ];
        sprintf ( buf, "$%X", memory->GetWord( pc + 2 ));
        PUSH( buf, 4 );
    }

    // (xxx).l
    if ( ( mode == 7 )&& ( reg == 1 ))
    {
        memory->SetLongword( memory->GetLongword( pc + 2 ), a[ 7 ].l );
        pc += 6;

        buf = new char[ 32 ];
        sprintf ( buf, "$%lX", memory->GetLongword( pc + 2 ));
        PUSH( buf, 4 );
    }
}

void M68008::rol_rReg()
{
    int cnt, dr, size, ir, reg;
    int data, mask;


    cnt  = GETBITS( pc, 0x0E00, 9 );
    dr   = GETBITS( pc, 0x0100, 8 );
    size = GETBITS( pc, 0x00C0, 6 );
    ir   = GETBITS( pc, 0x0020, 5 );
    reg  = GETBITS( pc, 0x0007, 0 );

    // rotate right
    if( dr == 0 )
    {
        // #data,dn
        if( ir == 0 )
        {
            if( !cnt ) cnt = 8;

            // byte
            if( size == 0 )
            {
                mask = IPOW( 2, cnt )- 1;
                data = d[ reg ].b & mask;
                ccr.c = data & IPOW( 2, cnt - 1 );
                d[ reg ].b = ( d[ reg ].b >> cnt ) | ( data << ( 8 - cnt ));
            }

            // word
            if( size == 1 )
```

```
        {
            mask = IPOW( 2, cnt )- 1;
            data = d[ reg ].w & mask;
            ccr.c = data & IPOW( 2, cnt - 1 );
            d[ reg ].w = ( d[ reg ].w >> cnt ) | ( data << ( 16 - cnt ));
        }

        // longword
        if( size == 2 )
        {
            mask = IPOW( 2, cnt )- 1;
            data = d[ reg ].l & mask;
            ccr.c = data & IPOW( 2, cnt - 1 );
            d[ reg ].l = ( d[ reg ].l >> cnt ) | ( data << ( 32 - cnt ));
        }
    }

    // dn,dn
    if( ir == 1 )
    {
        // byte
        if( size == 0 )
        {
            cnt = d[ cnt ].b % 8;
            mask = IPOW( 2, cnt )- 1;
            data = d[ reg ].b & mask;
            ccr.c = data & IPOW( 2, cnt - 1 );
            d[ reg ].b = ( d[ reg ].b >> cnt ) | ( data << ( 8 - cnt ));
        }

        // word
        if( size == 1 )
        {
            cnt = d[ cnt ].w % 16;
            mask = IPOW( 2, cnt )- 1;
            data = d[ reg ].w & mask;
            ccr.c = data & IPOW( 2, cnt - 1 );
            d[ reg ].w = ( d[ reg ].w >> cnt ) | ( data << ( 16 - cnt ));
        }

        // longword
        if( size == 2 )
        {
            cnt = d[ cnt ].l % 32;
            mask = IPOW( 2, cnt )- 1;
            data = d[ reg ].l & mask;
            ccr.c = data & IPOW( 2, cnt - 1 );
            d[ reg ].l = ( d[ reg ].l >> cnt ) | ( data << ( 32 - cnt ));
        }
    }
}
```

```
// rotate  left
if( dr == 1 )
{
   // #data,dn
   if( ir == 0 )
   {
      if( !cnt ) cnt = 8;

      // byte
      if( size == 0 )
      {
         mask = ( IPOW( 2, cnt )- 1 )<< ( 8 - cnt );
         data = ( d[ reg ].b & mask )>> ( 8 - cnt );
         ccr.c = data & 1;
         d[ reg ].b = ( d[ reg ].b << cnt ) | data;
      }

      // word
      if( size == 1 )
      {
         mask = ( IPOW( 2, cnt )- 1 )<< ( 16 - cnt );
         data = ( d[ reg ].w & mask )>> ( 16 - cnt );
         ccr.c = data & 1;
         d[ reg ].w = ( d[ reg ].w << cnt )| data;
      }

      // longword
      if( size == 2 )
      {
         mask = ( IPOW( 2, cnt )- 1 )<< ( 32 - cnt );
         data = ( d[ reg ].l & mask )>> ( 32 - cnt );
         ccr.c = data & 1;
         d[ reg ].l = ( d[ reg ].l << cnt ) | data;
      }
   }

   // dn,dn
   if( ir == 1 )
   {
      // byte
      if( size == 0 )
      {
         cnt = d[ cnt ].b % 8;
         mask = ( IPOW( 2, cnt )- 1 )<< ( 8 - cnt );
         data = ( d[ reg ].b & mask )>> ( 8 - cnt );
         ccr.c = data & 1;
         d[ reg ].b = ( d[ reg ].b << cnt ) | data;
      }

      // word
      if( size == 1 )
      {
```

```
            cnt = d[ cnt ].w % 16;
            mask = ( IPOW( 2, cnt )- 1 )<< ( 16 - cnt );
            data = ( d[ reg ].w & mask )>> ( 16 - cnt );
            ccr.c = data & 1;
            d[ reg ].w = ( d[ reg ].w << cnt )| data;
         }

         // longword
         if( size == 2 )
         {
            cnt = d[ cnt ].l % 32;
            mask = ( IPOW( 2, cnt )- 1 )<< ( 32 - cnt );
            data = ( d[ reg ].l & mask )>> ( 32 - cnt );
            ccr.c = data & 1;
            d[ reg ].l = ( d[ reg ].l << cnt ) | data;
         }
      }
   }

   pc += 2;
}

void M68008::rol_rMem()
{
   int mode, reg, dr, xn;
   unsigned long displ;

   dr   = GETBITS( pc, 0x0100, 8 );
   mode = GETBITS( pc, 0x0038, 3 );
   reg  = GETBITS( pc, 0x0007, 0 );

   ccr.v = 0;

   // (an)
   if( mode == 2 )
   {
      // right
      if( dr == 0 )
      {
         ccr.c = memory->GetWord( a[ reg ].l )& 1;
         memory->SetWord( memory->GetWord( a[ reg ].l )>> 1, a[ reg ].l );
         memory->SetWord( memory->GetWord( a[ reg ].l )| ( ccr.c << 15 ), a[ reg ].l );
         SETNZW( memory->GetWord( a[ reg ].l ));
      }

      // left
      if( dr == 1 )
      {
         ccr.c = ( memory->GetWord( a[ reg ].l )& MSB16 )>> 15;
         memory->SetWord( memory->GetWord( a[ reg ].l )<< 1, a[ reg ].l );
         memory->SetWord( memory->GetWord( a[ reg ].l )| ccr.c, a[ reg ].l );
         SETNZW( memory->GetWord( a[ reg ].l ));
```

```
    }

    pc += 2;
}

// (an)+
if( mode == 3 )
{
    // right
    if( dr == 0 )
    {
        ccr.c = memory->GetWord( a[ reg ].l )& 1;
        memory->SetWord( memory->GetWord( a[ reg ].l )>> 1, a[ reg ].l );
        memory->SetWord( memory->GetWord( a[ reg ].l )| ( ccr.c << 15 ), a[ reg ].l );
        SETNZW( memory->GetWord( a[ reg ].l ));
        a[ reg ].l += 2;
    }

    // left
    if( dr == 1 )
    {
        ccr.c = ( memory->GetWord( a[ reg ].l )& MSB16 )>> 15;
        memory->SetWord( memory->GetWord( a[ reg ].l )<< 1, a[ reg ].l );
        memory->SetWord( memory->GetWord( a[ reg ].l )| ccr.c, a[ reg ].l );
        SETNZW( memory->GetWord( a[ reg ].l ));
        a[ reg ].l += 2;
    }

    pc += 2;
}

// -(an)
if( mode == 4 )
{
    // right
    if( dr == 0 )
    {
        a[ reg ].l -= 2;
        ccr.c = memory->GetWord( a[ reg ].l )& 1;
        memory->SetWord( memory->GetWord( a[ reg ].l )>> 1, a[ reg ].l );
        memory->SetWord( memory->GetWord( a[ reg ].l )| ( ccr.c << 15 ), a[ reg ].l );
        SETNZW( memory->GetWord( a[ reg ].l ));
    }

    // left
    if( dr == 1 )
    {
        a[ reg ].l -= 2;
        ccr.c = ( memory->GetWord( a[ reg ].l )& MSB16 )>> 15;
        memory->SetWord( memory->GetWord( a[ reg ].l )<< 1, a[ reg ].l );
        memory->SetWord( memory->GetWord( a[ reg ].l )| ccr.c, a[ reg ].l );
        SETNZW( memory->GetWord( a[ reg ].l ));
```

```
    }

    pc += 2;
}

// d16(an)
if( mode == 5 )
{
    displ = memory->GetWord( pc + 2 )+ a[ reg ].l;

    // right
    if( dr == 0 )
    {
        ccr.c = memory->GetWord( displ )& 1;
        memory->SetWord( memory->GetWord( displ )>> 1, displ );
        memory->SetWord( memory->GetWord( displ )| ( ccr.c << 15 ), displ );
        SETNZW( memory->GetWord( displ ));
    }

    // left
    if( dr == 1 )
    {
        ccr.c = ( memory->GetWord( displ )& MSB16 )>> 15;
        memory->SetWord( memory->GetWord( displ )<< 1, displ );
        memory->SetWord( memory->GetWord( displ )| ccr.c, displ );
        SETNZW( memory->GetWord( displ ));
    }

    pc += 4;
}

// d8(an,xn)
if( mode == 6 )
{
    xn = GETBITS( pc + 2, 0xF000, 12 );
    displ = memory->GetByte( pc + 3 )+ a[ reg ].l + d[ xn ].l;

    // right
    if( dr == 0 )
    {
        ccr.c = memory->GetWord( displ )& 1;
        memory->SetWord( memory->GetWord( displ )>> 1, displ );
        memory->SetWord( memory->GetWord( displ )| ( ccr.c << 15 ), displ );
        SETNZW( memory->GetWord( displ ));
    }

    // left
    if( dr == 1 )
    {
        ccr.c = ( memory->GetWord( displ )& MSB16 )>> 15;
        memory->SetWord( memory->GetWord( displ )<< 1, displ );
        memory->SetWord( memory->GetWord( displ )| ccr.c, displ );
```

```
            SETNZW( memory->GetWord( displ ));
        }

        pc += 4;
    }

    if( mode == 7 )
    {
        // (xxx).w
        if( reg == 0 )
        {
            displ = memory->GetWord( pc + 2 );
            pc += 4;
        }
        else
        {
            displ = memory->GetLongword( pc + 2 );
            pc += 6;
        }

        // right
        if( dr == 0 )
        {
            ccr.c = memory->GetWord( displ )& 1;
            memory->SetWord( memory->GetWord( displ )>> 1, displ );
            memory->SetWord( memory->GetWord( displ )| ( ccr.c << 15 ), displ );
            SETNZW( memory->GetWord( displ ));
        }

        // left
        if( dr == 1 )
        {
            ccr.c = ( memory->GetWord( displ )& MSB16 )>> 15;
            memory->SetWord( memory->GetWord( displ )<< 1, displ );
            memory->SetWord( memory->GetWord( displ )| ccr.c, displ );
            SETNZW( memory->GetWord( displ ));
        }
    }
}

void M68008::roxl_rReg()
{
    int cnt, dr, size, ir, reg, oldx;
    int mask, data;


    cnt  = GETBITS( pc, 0x0E00, 9 );
    dr   = GETBITS( pc, 0x0100, 8 );
    size = GETBITS( pc, 0x00C0, 6 );
    ir   = GETBITS( pc, 0x0020, 5 );
    reg  = GETBITS( pc, 0x0007, 0 );
```

```
// right
if( dr == 0 )
{
    // #data
    if( ir == 0 )
    {
        // byte
        if( size == 0 )
        {
            mask = ( IPOW( 2, cnt )- 1 );
            data = ( d[ reg ].b & mask )<< ( 8 - cnt );
            oldx = ccr.x;
            ccr.c = ccr.x = ( data & MSB8 )? 1 : 0;
            data <<= 1;
            d[ reg ].b = ( d[ reg ].b >> cnt ) | data | ( oldx << ( 8 - cnt ) );
        }

        // word
        if( size == 1 )
        {
            if( !cnt ) cnt = 8;
            mask = ( IPOW( 2, cnt )- 1 );
            data = ( d[ reg ].w & mask )<< ( 16 - cnt );
            oldx = ccr.x;
            ccr.c = ccr.x = ( data & MSB16 )? 1 : 0;
            data <<= 1;
            d[ reg ].w = ( d[ reg ].w >> cnt )| data | ( oldx << ( 16 - cnt ) );
        }

        // longword
        if( size == 2 )
        {
            if( !cnt ) cnt = 8;
            mask = ( IPOW( 2, cnt )- 1 );
            data = ( d[ reg ].l & mask )<< ( 32 - cnt );
            oldx = ccr.x;
            ccr.c = ccr.x = ( data & MSB32 )? 1 : 0;
            data <<= 1;
            d[ reg ].l = ( d[ reg ].l >> cnt ) | data | ( oldx << ( 32 - cnt ) );
        }
    }

    // dn
    if( ir == 1 )
    {
        // byte
        if( size == 0 )
        {
            cnt = d[ cnt ].b % 8;
            mask = ( IPOW( 2, cnt )- 1 );
            data = ( d[ reg ].b & mask )<< ( 8 - cnt );
            oldx = ccr.x;
```

```
            ccr.c = ccr.x = ( data & MSB8 )? 1 : 0;
            data <<= 1;
            d[ reg ].b = ( d[ reg ].b >> cnt ) | data | ( oldx << ( 8 - cnt ) );
        }

        // word
        if( size == 1 )
        {
            cnt = d[ cnt ].w % 16;
            mask = ( IPOW( 2, cnt )- 1 );
            data = ( d[ reg ].w & mask )<< ( 16 - cnt );
            oldx = ccr.x;
            ccr.c = ccr.x = ( data & MSB16 )? 1 : 0;
            data <<= 1;
            d[ reg ].w = ( d[ reg ].w >> cnt )| data | ( oldx << ( 16 - cnt ) );
        }

        // longword
        if( size == 2 )
        {
            cnt = d[ cnt ].l % 32;
            mask = ( IPOW( 2, cnt )- 1 );
            data = ( d[ reg ].l & mask )<< ( 32 - cnt );
            oldx = ccr.x;
            ccr.c = ccr.x = ( data & MSB32 )? 1 : 0;
            data <<= 1;
            d[ reg ].l = ( d[ reg ].l >> cnt ) | data | ( oldx << ( 32 - cnt ) );
        }
    }
}

// left
if( dr == 1 )
{
    // #data
    if( ir == 0 )
    {
        // byte
        if( size == 0 )
        {
            mask = ( IPOW( 2, cnt )- 1 )<< ( 8 - cnt );
            data = ( d[ reg ].b & mask )>> ( 8 - cnt );
            oldx = ccr.x;
            ccr.x = data & 1;
            data >>= 1;
            d[ reg ].b = ( d[ reg ].b << cnt ) | data | ( oldx << ( cnt - 1 ) );
        }

        // word
        if( size == 1 )
        {
            if( !cnt ) cnt = 8;
```

```
            mask = ( IPOW( 2, cnt )- 1 )<< ( 16 - cnt );
            data = ( d[ reg ].w & mask )>> ( 16 - cnt );
            oldx = ccr.x;
            ccr.x = data & 1;
            data >>= 1;
            d[ reg ].w = ( d[ reg ].w << cnt )| data | ( oldx << ( cnt - 1 ) );
        }

        // longword
        if( size == 2 )
        {
            if( !cnt ) cnt = 8;
            mask = ( IPOW( 2, cnt )- 1 )<< ( 32 - cnt );
            data = ( d[ reg ].l & mask )>> ( 32 - cnt );
            oldx = ccr.x;
            ccr.x = data & 1;
            data >>= 1;
            d[ reg ].l = ( d[ reg ].l << cnt ) | data | ( oldx << ( cnt - 1 ) );
        }
    }

    // dn
    if( ir == 1 )
    {
        // byte
        if( size == 0 )
        {
            cnt = d[ cnt ].b % 8;
            mask = ( IPOW( 2, cnt )- 1 )<< ( 8 - cnt );
            data = ( d[ reg ].b & mask )>> ( 8 - cnt );
            oldx = ccr.x;
            ccr.x = data & 1;
            data >>= 1;
            d[ reg ].b = ( d[ reg ].b << cnt ) | data | ( oldx << ( cnt - 1 ) );
        }

        // word
        if( size == 1 )
        {
            cnt = d[ cnt ].w % 16;
            mask = ( IPOW( 2, cnt )- 1 )<< ( 16 - cnt );
            data = ( d[ reg ].w & mask )>> ( 16 - cnt );
            oldx = ccr.x;
            ccr.x = data & 1;
            data >>= 1;
            d[ reg ].w = ( d[ reg ].w << cnt )| data | ( oldx << ( cnt - 1 ) );
        }

        // longword
        if( size == 2 )
        {
            cnt = d[ cnt ].l % 32;
```

```
            mask = ( IPOW( 2, cnt )- 1 )<< ( 32 - cnt );
            data = ( d[ reg ]. l & mask )>> ( 32 - cnt );
            oldx = ccr.x;
            ccr.x = data & 1;
            data >>= 1;
            d[ reg ]. l = ( d[ reg ]. l << cnt ) | data | ( oldx << ( cnt - 1 ) );
         }
      }
   }

   pc += 2;
}

void M68008::roxl_rMem()
{
   int mode, reg, dr, xn;
   unsigned long displ;

   dr   = GETBITS( pc, 0x0100, 8 );
   mode = GETBITS( pc, 0x0038, 3 );
   reg  = GETBITS( pc, 0x0007, 0 );

   ccr.v = 0;

   // (an)
   if( mode == 2 )
   {
      // right
      if( dr == 0 )
      {
         ccr.c = memory->GetWord( a[ reg ].l )& 1;
         memory->SetWord( ( memory->GetWord( a[ reg ].l )>> 1 )| ( ccr.x << 15 ), a[ reg ].l );
         ccr.x = ccr.c;
         SETNZW( memory->GetWord( a[ reg ].l ));
      }

      // left
      if( dr == 1 )
      {
         ccr.c = ( memory->GetWord( a[ reg ].l )& MSB16 )>> 15;
         memory->SetWord( ( memory->GetWord( a[ reg ].l )<< 1 )| ccr.x, a[ reg ].l );
         ccr.x = ccr.c;
         SETNZW( memory->GetWord( a[ reg ].l ));
      }

      pc += 2;
   }

   // (an)+
   if( mode == 3 )
   {
      // right
```

```
    if( dr == 0 )
    {
        ccr.c = memory->GetWord( a[ reg ].l )& 1;
        memory->SetWord( ( memory->GetWord( a[ reg ].l )>> 1 )| ( ccr.x << 15 ), a[ reg ].l );
        ccr.x = ccr.c;
        SETNZW( memory->GetWord( a[ reg ].l ));
        a[ reg ].l += 2;
    }

    // left
    if( dr == 1 )
    {
        ccr.c = ( memory->GetWord( a[ reg ].l )& MSB16 )>> 15;
        memory->SetWord( ( memory->GetWord( a[ reg ].l )<< 1 )| ccr.x, a[ reg ].l );
        ccr.x = ccr.c;
        SETNZW( memory->GetWord( a[ reg ].l ));
        a[ reg ].l += 2;
    }

    pc += 2;
}

// -(an)
if( mode == 4 )
{
    // right
    if( dr == 0 )
    {
        a[ reg ].l -= 2;
        ccr.c = memory->GetWord( a[ reg ].l )& 1;
        memory->SetWord( ( memory->GetWord( a[ reg ].l )>> 1 )| ( ccr.x << 15 ), a[ reg ].l );
        ccr.x = ccr.c;
        SETNZW( memory->GetWord( a[ reg ].l ));
        a[ reg ].l += 2;
    }

    // left
    if( dr == 1 )
    {
        a[ reg ].l -= 2;
        ccr.c = ( memory->GetWord( a[ reg ].l )& MSB16 )>> 15;
        memory->SetWord( ( memory->GetWord( a[ reg ].l )<< 1 )| ccr.x, a[ reg ].l );
        ccr.x = ccr.c;
        SETNZW( memory->GetWord( a[ reg ].l ));
    }

    pc += 2;
}

// d16(an)
if( mode == 5 )
{
```

```
    displ = memory->GetWord( pc + 2 )+ a[ reg ].l;

    // right
    if ( dr == 0 )
    {
        ccr.c = memory->GetWord( displ )& 1;
        memory->SetWord( ( memory->GetWord( displ )>> 1 )| ( ccr.x << 15 ), displ );
        ccr.x = ccr.c;
        SETNZW( memory->GetWord( displ ));
    }

    // left
    if ( dr == 1 )
    {
        ccr.c = ( memory->GetWord( displ )& MSB16 )>> 15;
        memory->SetWord( ( memory->GetWord( displ )<< 1 )| ccr.x, displ );
        ccr.x = ccr.c;
        SETNZW( memory->GetWord( displ ));
    }

    pc += 4;
}

// d8(an,xn)
if ( mode == 6 )
{
    xn = GETBITS( pc + 2, 0xF000, 12 );
    displ = memory->GetByte( pc + 3 )+ a[ reg ].l + d[ xn ].l;

    // right
    if ( dr == 0 )
    {
        ccr.c = memory->GetWord( displ )& 1;
        memory->SetWord( ( memory->GetWord( displ )>> 1 )| ( ccr.x << 15 ), displ );
        ccr.x = ccr.c;
        SETNZW( memory->GetWord( displ ));
    }

    // left
    if ( dr == 1 )
    {
        ccr.c = ( memory->GetWord( displ )& MSB16 )>> 15;
        memory->SetWord( ( memory->GetWord( displ )<< 1 )| ccr.x, displ );
        ccr.x = ccr.c;
        SETNZW( memory->GetWord( displ ));
    }

    pc += 4;
}

if ( mode == 7 )
{
```

```
        // (xxx).w
        if( reg == 0 )
        {
            displ = memory->GetWord( pc + 2 );
            pc += 4;
        }
        else
        {
            displ = memory->GetLongword( pc + 2 );
            pc += 6;
        }

        // right
        if( dr == 0 )
        {
            ccr.c = memory->GetWord( displ )& 1;
            memory->SetWord( ( memory->GetWord( displ )>> 1 )| ( ccr.x << 15 ), displ );
            ccr.x = ccr.c;
            SETNZW( memory->GetWord( displ ));
        }

        // left
        if( dr == 1 )
        {
            ccr.c = ( memory->GetWord( displ )& MSB16 )>> 15;
            memory->SetWord( ( memory->GetWord( displ )<< 1 )| ccr.x, displ );
            ccr.x = ccr.c;
            SETNZW( memory->GetWord( displ ));
        }
    }
}

void M68008::rts()
{

    char *buf;
    unsigned long int oldPC = pc;

    pc = memory->GetLongword( a[7].l );

    if( systemStack.getCount() <= 0 )
    {
        buf = new char[ 128 ];
        sprintf( buf, "ERROR at address $%lX: You are trying to pop the return address off an empty system
            stack.", oldPC );
        helpStack.push( buf );
    }
    else
    {
        POP( 4 );

        if( pushPop.getTop() != a[ 7 ].l )
```

```
        {
            buf = new char[ 128 ];
            sprintf ( buf, "WARNING at address $%lX: It seems that the number of pushes may not\n equal
                the numbof pops in the subroutine you just left.", oldPC );
            helpStack.push( buf );
        }
        pushPop.pop();
    }
    if ( !pushStack.pop() )
    {
        buf = new char[ 128 ];
        sprintf ( buf, "WARNING at address $%lX: You haven't used the stack at all in this\nsubroutine. You
            may have caused unwanted side effects.", oldPC );
        helpStack.push( buf );
    }

    a[7].l += 4;

}

void M68008::sbcd()
{
}

void M68008::sxx()
{

    int cond, mode, reg, xn;
    unsigned long displ = 0;
    unsigned char result;

    cond = GETBITS( pc, 0x0F00, 8 );
    mode = GETBITS( pc, 0x0038, 3 );
    reg  = GETBITS( pc, 0x0007, 0 );

    if( cond == 0 ) result = 0xFF;
    if( cond == 2 ) if( !ccr.c && !ccr.z ) result = 0xFF;
    if( cond == 3 ) if( ccr.c || ccr.z ) result = 0xFF;
    if( cond == 4 ) if( ccr.c ) result = 0xFF;
    if( cond == 5 ) if( ccr.c ) result = 0xFF;
    if( cond == 6 ) if( ccr.z ) result = 0xFF;
    if( cond == 7 ) if( ccr.z ) result = 0xFF;
    if( cond == 8 ) if( ccr.v ) result = 0xFF;
    if( cond == 9 ) if( ccr.v ) result = 0xFF;
    if( cond == 10 )if( ccr.n ) result = 0xFF;
    if( cond == 11 )if( ccr.n ) result = 0xFF;
    if( cond == 12 )if( ccr.n && ccr.v || !ccr.n && !ccr.v ) result = 0xFF;
    if( cond == 13 )if( ccr.n && !ccr.v || !ccr.n && ccr.v ) result = 0xFF;
    if( cond == 14 )if( ccr.n && ccr.v && !ccr.z || !ccr.n && !ccr.v && !ccr.z ) result = 0xFF;
    if( cond == 15 )if( ccr.z || ccr.n && !ccr.v || !ccr.n && ccr.v ) result = 0xFF;

    // dn
```

```
if( mode == 0 )
{
    d[ reg ].b = result;
    pc += 2;
}

// (an)
if( mode == 2 )
{
    memory->SetByte( result, a[ reg ].l );
    pc += 2;
}

// (an)+
if( mode == 3 )
{
    memory->SetByte( result, a[ reg ].l );
    a[ reg ].l += 1;
    pc += 2;
}

// -(an)
if( mode == 4 )
{
    a[ reg ].l -= 1;
    memory->SetByte( result, a[ reg ].l );
    pc += 2;
}

// d16(an)
if( mode == 5 )
{
    displ = memory->GetWord( pc + 2 )+ a[ reg ].l;
    memory->SetByte( result, displ );
    pc += 4;
}

// d8(an,xn)
if( mode == 6 )
{
    xn = GETBITS( pc + 2, 0xF000, 12 );
    displ = memory->GetByte( pc + 3 )+ a[ reg ].l + d[ xn ].l;
    memory->SetByte( result, displ );
    pc += 4;
}

if( mode == 7 )
{
    // (xxx).w
    if( reg == 0 )
    {
        displ = memory->GetWord( pc + 2 );
```

```
            pc += 4;
        }

        // (xxx).l
        if( reg == 1 )
        {
            displ = memory->GetLongword( pc + 2 );
            pc += 6;
        }

        memory->SetByte( result, displ );
    }
}


void M68008::sub()
{
    int mode, opmode, reg, eaReg;
    unsigned long addr;
    DATA_REGISTER( result );


    reg    = GETBITS( pc, 0x0E00, 9 );
    opmode = GETBITS( pc, 0x01C0, 6 );
    mode   = GETBITS( pc, 0x0038, 3 );
    eaReg  = GETBITS( pc, 0x0007, 0 );

    if( ( opmode & 3 )== 3 )
    {
        suba();
        return;
    }

    // dn,dn
    if( mode == 0 )
    {
        if( opmode == 0 )
        {
            result.b = d[ reg ].b - d[ eaReg ].b;
            SUB_SETC( d[ eaReg ].b, d[ reg ].b, result.b, MSB8 );
            SUB_SETV( d[ eaReg ].b, d[ reg ].b, result.b, MSB8 );
            d[ reg ].b = result.b;
            SETNZB( d[ reg ].b );
        }
        if( opmode == 1 )
        {
            result.w = d[ reg ].w - d[ eaReg ].w;
            SUB_SETC( d[ eaReg ].w, d[ reg ].w, result.w, MSB16 );
            SUB_SETV( d[ eaReg ].w, d[ reg ].w, result.w, MSB16 );
            d[ reg ].w = result.w;
            SETNZW( d[ reg ].w );
        }
```

```
    if ( opmode == 2 )
    {
        result.l = d[ reg ].l - d[ eaReg ].l;
        SUB_SETC( d[ eaReg ].l, d[ reg ].l, result.l, MSB32 );
        SUB_SETV( d[ eaReg ].l, d[ reg ].l, result.l, MSB32 );
        d[ reg ].l = result.l;
        SETNZL( d[ reg ].l );
    }
    if ( opmode == 4 )
    {
        result.b = d[ eaReg ].b - d[ reg ].b;
        SUB_SETC( d[ reg ].b, d[ eaReg ].b, result.b, MSB8 );
        SUB_SETV( d[ reg ].b, d[ eaReg ].b, result.b, MSB8 );
        d[ eaReg ].b = result.b;
        SETNZB( d[ eaReg ].b );
    }
    if ( opmode == 5 )
    {
        result.w = d[ eaReg ].w - d[ reg ].w;
        SUB_SETC( d[ reg ].w, d[ eaReg ].w, result.w, MSB16 );
        SUB_SETV( d[ reg ].w, d[ eaReg ].w, result.w, MSB16 );
        d[ eaReg ].w = result.w;
        SETNZW( d[ eaReg ].w );
    }
    if ( opmode == 6 )
    {
        result.l = d[ eaReg ].l - d[ reg ].l;
        SUB_SETC( d[ reg ].l, d[ eaReg ].l, result.l, MSB32 );
        SUB_SETV( d[ reg ].l, d[ eaReg ].l, result.l, MSB32 );
        d[ eaReg ].l = result.l;
        SETNZL( d[ eaReg ].l );
    }
    pc += 2;
}

// an,dn ( dn - an )
if ( mode == 1 )
{
    if ( opmode == 1 )
    {
        result.w = d[ reg ].w - a[ eaReg ].w;
        SETNZW( result.w );
        SUB_SETC( a[ eaReg ].w, d[ reg ].w, result.w, MSB16 );
        SUB_SETV( a[ eaReg ].w, d[ reg ].w, result.w, MSB16 );
        d[ reg ].w = result.w;
    }
    if ( opmode == 2 )
    {
        result.l = d[ reg ].l - a[ eaReg ].l;
        SETNZL( result.l );
        SUB_SETC( a[ eaReg ].l, d[ reg ].l, result.l, MSB32 );
        SUB_SETV( a[ eaReg ].l, d[ reg ].l, result.l, MSB32 );
```

```
        d[ reg ].l = result.l;
    }
    pc += 2;
}

// (an),dn ( dn - (an) )
if( mode == 2 )
{
    if( opmode == 0 )
    {
        result.b = d[ reg ].b - memory->GetByte( a[ eaReg ].l );
        SETNZB( result.b );
        SUB_SETC( memory->GetByte( a[ eaReg ].l ), d[ reg ].b, result.b, MSB8 );
        SUB_SETV( memory->GetByte( a[ eaReg ].l ), d[ reg ].b, result.b, MSB8 );
        d[ reg ].b = result.b;
    }
    if( opmode == 1 )
    {
        result.w = d[ reg ].w - memory->GetWord( a[ eaReg ].l );
        SETNZW( result.w );
        SUB_SETC( memory->GetWord( a[ eaReg ].l ), d[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( a[ eaReg ].l ), d[ reg ].w, result.w, MSB16 );
        d[ reg ].w = result.w;
    }
    if( opmode == 2 )
    {
        result.l = d[ reg ].l - memory->GetLongword( a[ eaReg ].l );
        SETNZL( result.l );
        SUB_SETC( memory->GetLongword( a[ eaReg ].l ), d[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( a[ eaReg ].l ), d[ reg ].l, result.l, MSB32 );
        d[ reg ].l = result.l;
    }

    if( opmode == 4 )
    {
        result.b = memory->GetByte( a[ eaReg ].l )- d[ reg ].b;
        SETNZB( result.b );
        SUB_SETC( d[ reg ].b, memory->GetByte( a[ eaReg ].l ), result.b, MSB8 );
        SUB_SETV( d[ reg ].b, memory->GetByte( a[ eaReg ].l ), result.b, MSB8 );
        memory->SetByte( result.b, a[ eaReg ].l );
    }
    if( opmode == 5 )
    {
        result.w = memory->GetWord( a[ eaReg ].l )- d[ reg ].w;
        SETNZW( result.w );
        SUB_SETC( d[ reg ].w, memory->GetWord( a[ eaReg ].l ), result.w, MSB16 );
        SUB_SETV( d[ reg ].w, memory->GetWord( a[ eaReg ].l ), result.w, MSB16 );
        memory->SetWord( result.w, a[ eaReg ].l );
    }
    if( opmode == 6 )
    {
        result.l = memory->GetLongword( a[ eaReg ].l )- d[ reg ].l;
```

```
            SETNZL( result.l );
            SUB_SETC( d[ reg ].l, memory->GetLongword( a[ eaReg ].l ), result.l, MSB32 );
            SUB_SETV( d[ reg ].l, memory->GetLongword( a[ eaReg ].l ), result.l, MSB32 );
            memory->SetLongword( result.l, a[ eaReg ].l );
        }

        pc += 2;
    }

    // (an)+,dn ( dn - (an) )
    if( mode == 3 )
    {
        if( opmode == 0 )
        {
            result.b = d[ reg ].b - memory->GetByte( a[ eaReg ].l );
            SETNZB( result.b );
            SUB_SETC( memory->GetByte( a[ eaReg ].l ), d[ reg ].b, result.b, MSB8 );
            SUB_SETV( memory->GetByte( a[ eaReg ].l ), d[ reg ].b, result.b, MSB8 );
            d[ reg ].b = result.b;
            a[ eaReg ].l += 1;
        }
        if( opmode == 1 )
        {
            result.w = d[ reg ].w - memory->GetWord( a[ eaReg ].l );
            SETNZW( result.w );
            SUB_SETC( memory->GetWord( a[ eaReg ].l ), d[ reg ].w, result.w, MSB16 );
            SUB_SETV( memory->GetWord( a[ eaReg ].l ), d[ reg ].w, result.w, MSB16 );
            d[ reg ].w = result.w;
            a[ eaReg ].l += 2;
        }
        if( opmode == 2 )
        {
            result.l = d[ reg ].l - memory->GetLongword( a[ eaReg ].l );
            SETNZL( result.l );
            SUB_SETC( memory->GetLongword( a[ eaReg ].l ), d[ reg ].l, result.l, MSB32 );
            SUB_SETV( memory->GetLongword( a[ eaReg ].l ), d[ reg ].l, result.l, MSB32 );
            d[ reg ].l = result.l;
            a[ eaReg ].l += 4;
        }
        if( opmode == 4 )
        {
            result.b = memory->GetByte( a[ eaReg ].l )- d[ reg ].b;
            SETNZB( result.b );
            SUB_SETC( d[ reg ].b, memory->GetByte( a[ eaReg ].l ), result.b, MSB8 );
            SUB_SETV( d[ reg ].b, memory->GetByte( a[ eaReg ].l ), result.b, MSB8 );
            memory->SetByte( result.b, a[ eaReg ].l );
            a[ eaReg ].l += 1;
        }
        if( opmode == 5 )
        {
            result.w = memory->GetWord( a[ eaReg ].l )- d[ reg ].w;
            SETNZW( result.w );
```

```
        SUB_SETC( d[ reg ].w, memory->GetWord( a[ eaReg ].l ), result.w, MSB16 );
        SUB_SETV( d[ reg ].w, memory->GetWord( a[ eaReg ].l ), result.w, MSB16 );
        memory->SetWord( result.w, a[ eaReg ].l );
        a[ eaReg ].l += 2;
    }
    if( opmode == 6 )
    {
        result.l = memory->GetLongword( a[ eaReg ].l )- d[ reg ].l;
        SETNZL( result.l );
        SUB_SETC( d[ reg ].l, memory->GetLongword( a[ eaReg ].l ), result.l, MSB32 );
        SUB_SETV( d[ reg ].l, memory->GetLongword( a[ eaReg ].l ), result.l, MSB32 );
        memory->SetLongword( result.l, a[ eaReg ].l );
        a[ eaReg ].l += 4;
    }

    pc += 2;
}

// -(an),dn ( dn - (an) )
if( mode == 4 )
{
    if( opmode == 0 )
    {
        a[ eaReg ].l -= 1;
        result.b = d[ reg ].b - memory->GetByte( a[ eaReg ].l );
        SETNZB( result.b );
        SUB_SETC( memory->GetByte( a[ eaReg ].l ), d[ reg ].b, result.b, MSB8 );
        SUB_SETV( memory->GetByte( a[ eaReg ].l ), d[ reg ].b, result.b, MSB8 );
        d[ reg ].b = result.b;
    }
    if( opmode == 1 )
    {
        a[ eaReg ].l -= 2;
        result.w = d[ reg ].w - memory->GetWord( a[ eaReg ].l );
        SETNZW( result.w );
        SUB_SETC( memory->GetWord( a[ eaReg ].l ), d[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( a[ eaReg ].l ), d[ reg ].w, result.w, MSB16 );
        d[ reg ].w = result.w;
    }
    if( opmode == 2 )
    {
        a[ eaReg ].l -= 4;
        result.l = d[ reg ].l - memory->GetLongword( a[ eaReg ].l );
        SETNZL( result.l );
        SUB_SETC( memory->GetLongword( a[ eaReg ].l ), d[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( a[ eaReg ].l ), d[ reg ].l, result.l, MSB32 );
        d[ reg ].l = result.l;
    }
    if( opmode == 4 )
    {
        a[ eaReg ].l -= 1;
        result.b = memory->GetByte( a[ eaReg ].l )- d[ reg ].b;
```

```
            SETNZB( result.b );
            SUB_SETC( d[ reg ].b, memory->GetByte( a[ eaReg ].l ), result.b, MSB8 );
            SUB_SETV( d[ reg ].b, memory->GetByte( a[ eaReg ].l ), result.b, MSB8 );
            memory->SetByte( result.b, a[ eaReg ].l );

        }
        if ( opmode == 5 )
        {
            a[ eaReg ].l -= 2;
            result.w = memory->GetWord( a[ eaReg ].l )- d[ reg ].w;
            SETNZW( result.w );
            SUB_SETC( d[ reg ].w, memory->GetWord( a[ eaReg ].l ), result.w, MSB16 );
            SUB_SETV( d[ reg ].w, memory->GetWord( a[ eaReg ].l ), result.w, MSB16 );
            memory->SetWord( result.w, a[ eaReg ].l );
        }
        if ( opmode == 6 )
        {
            a[ eaReg ].l -= 4;
            result.l = memory->GetLongword( a[ eaReg ].l )- d[ reg ].l;
            SETNZL( result.l );
            SUB_SETC( d[ reg ].l, memory->GetLongword( a[ eaReg ].l ), result.l, MSB32 );
            SUB_SETV( d[ reg ].l, memory->GetLongword( a[ eaReg ].l ), result.l, MSB32 );
            memory->SetLongword( result.l, a[ eaReg ].l );

        }

        pc += 2;

    }

    // d16(an),dn ( dn - d16(an) )
    if ( mode == 5 )
    {
        unsigned long addr = a[ eaReg ].l + (signed short int)memory->GetWord( pc + 2 );
        if ( opmode == 0 )
        {
            result.b = d[ reg ].b - memory->GetByte( addr );
            SETNZB( result.b );
            SUB_SETC( memory->GetByte( addr ), d[ reg ].b, result.b, MSB8 );
            SUB_SETV( memory->GetByte( addr ), d[ reg ].b, result.b, MSB8 );
            d[ reg ].b = result.b;
        }
        if ( opmode == 1 )
        {
            result.w = d[ reg ].w - memory->GetWord( addr );
            SETNZW( result.w );
            SUB_SETC( memory->GetWord( addr ), d[ reg ].w, result.w, MSB16 );
            SUB_SETV( memory->GetWord( addr ), d[ reg ].w, result.w, MSB16 );
            d[ reg ].w = result.w;
        }
        if ( opmode == 2 )
        {
```

```
        result.l = d[ reg ].l - memory->GetLongword( addr );
        SETNZL( result.l );
        SUB_SETC( memory->GetLongword( addr ), d[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( addr ), d[ reg ].l, result.l, MSB32 );
        d[ reg ].l = result.l;
    }
    if( opmode == 4 )
    {
        result.b = memory->GetByte( addr )- d[ reg ].b;
        SETNZB( result.b );
        SUB_SETC( d[ reg ].b, memory->GetByte( addr ), result.b, MSB8 );
        SUB_SETV( d[ reg ].b, memory->GetByte( addr ), result.b, MSB8 );
        memory->SetByte( result.b, addr );
    }
    if( opmode == 5 )
    {
        result.w = memory->GetWord( addr )- d[ reg ].w;
        SETNZW( result.w );
        SUB_SETC( d[ reg ].w, memory->GetWord( addr ), result.w, MSB16 );
        SUB_SETV( d[ reg ].w, memory->GetWord( addr ), result.w, MSB16 );
        memory->SetWord( result.w, addr );
    }
    if( opmode == 6 )
    {
        result.l = memory->GetLongword( addr )- d[ reg ].l;
        SETNZL( result.l );
        SUB_SETC( d[ reg ].l, memory->GetLongword( addr ), result.l, MSB32 );
        SUB_SETV( d[ reg ].l, memory->GetLongword( addr ), result.l, MSB32 );
        memory->SetLongword( result.l, addr );
    }
    pc += 4;
}

// d8(an,Xn),dn ( dn - d8(an) )
if( mode == 6 )
{
    addr = a[ eaReg ].l + (signed char)memory->GetByte( pc + 3 )
        + d[ GETBITS( pc + 2, 0xF000, 12 )].l;

    if( opmode == 0 )
    {
        result.b = d[ reg ].b - memory->GetByte( addr );
        SETNZB( result.b );
        SUB_SETC( memory->GetByte( addr ), d[ reg ].b, result.b, MSB8 );
        SUB_SETV( memory->GetByte( addr ), d[ reg ].b, result.b, MSB8 );
        d[ reg ].b = result.b;
    }
    if( opmode == 1 )
    {
        result.w = d[ reg ].w - memory->GetWord( addr );
        SETNZW( result.w );
        SUB_SETC( memory->GetWord( addr ), d[ reg ].w, result.w, MSB16 );
```

```
        SUB_SETV( memory->GetWord( addr ), d[ reg ].w, result.w, MSB16 );
        d[ reg ].w = result.w;
    }
    if( opmode == 2 )
    {
        result.l = d[ reg ].l - memory->GetLongword( addr );
        SETNZL( result.l );
        SUB_SETC( memory->GetLongword( addr ), d[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( addr ), d[ reg ].l, result.l, MSB32 );
        d[ reg ].l = result.l;
    }
    if( opmode == 4 )
    {
        result.b = memory->GetByte( addr )- d[ reg ].b;
        SETNZB( result.b );
        SUB_SETC( d[ reg ].b, memory->GetByte( addr ), result.b, MSB8 );
        SUB_SETV( d[ reg ].b, memory->GetByte( addr ), result.b, MSB8 );
        memory->SetByte( result.b, addr );
    }
    if( opmode == 5 )
    {
        result.w = memory->GetWord( addr )- d[ reg ].w;
        SETNZW( result.w );
        SUB_SETC( d[ reg ].w, memory->GetWord( addr ), result.w, MSB16 );
        SUB_SETV( d[ reg ].w, memory->GetWord( addr ), result.w, MSB16 );
        memory->SetWord( result.w, addr );
    }
    if( opmode == 6 )
    {
        result.l = memory->GetLongword( addr )- d[ reg ].l;
        SETNZL( result.l );
        SUB_SETC( d[ reg ].l, memory->GetLongword( addr ), result.l, MSB32 );
        SUB_SETV( d[ reg ].l, memory->GetLongword( addr ), result.l, MSB32 );
        memory->SetLongword( result.l, addr );
    }
    pc += 4;
}

//(XXX).W,dn ( dn - (XXX).W - dn) )
if( mode == 7 )
{
    if( eaReg == 0 )
    {
        addr = memory->GetWord( pc + 2 );
        pc += 4;
    }
    else
    {
        addr = memory->GetLongword( pc + 2 );
        pc += 6;
    }
    if( opmode == 0 )
```

```
    {
        result.b = d[ reg ].b - memory->GetByte( addr );
        SETNZB( result.b );
        SUB_SETC( memory->GetByte( addr ), d[ reg ].b, result.b, MSB8 );
        SUB_SETV( memory->GetByte( addr ), d[ reg ].b, result.b, MSB8 );
        d[ reg ].b = result.b;
    }
    if( opmode == 1 )
    {
        result.w = d[ reg ].w - memory->GetWord( addr );
        SETNZW( result.w );
        SUB_SETC( memory->GetWord( addr ), d[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( addr ), d[ reg ].w, result.w, MSB16 );
        d[ reg ].w = result.w;
    }
    if( opmode == 2 )
    {
        result.l = d[ reg ].l - memory->GetLongword( addr );
        SETNZL( result.l );
        SUB_SETC( memory->GetLongword( addr ), d[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( addr ), d[ reg ].l, result.l, MSB32 );
        d[ reg ].l = result.l;
    }
    if( opmode == 4 )
    {
        result.b = memory->GetByte( addr )- d[ reg ].b;
        SETNZB( result.b );
        SUB_SETC( d[ reg ].b, memory->GetByte( addr ), result.b, MSB8 );
        SUB_SETV( d[ reg ].b, memory->GetByte( addr ), result.b, MSB8 );
        memory->SetByte( result.b, addr );
    }
    if( opmode == 5 )
    {
        result.w = memory->GetWord( addr )- d[ reg ].w;
        SETNZW( result.w );
        SUB_SETC( d[ reg ].w, memory->GetWord( addr ), result.w, MSB16 );
        SUB_SETV( d[ reg ].w, memory->GetWord( addr ), result.w, MSB16 );
        memory->SetWord( result.w, addr );
    }
    if( opmode == 6 )
    {
        result.l = memory->GetLongword( addr )- d[ reg ].l;
        SETNZL( result.l );
        SUB_SETC( d[ reg ].l, memory->GetLongword( addr ), result.l, MSB32 );
        SUB_SETV( d[ reg ].l, memory->GetLongword( addr ), result.l, MSB32 );
        memory->SetByte( result.b, addr );
    }
}

// OMITTED
// #< data >, dn ( see subi )
// ( D16, PC )
```

```
    // ( d8, PC, Xn )
}

void M68008::suba()
{
    int mode, opmode, reg, eaReg;
    unsigned long addr;
    ADDRESS_REGISTER( result );
    ADDRESS_REGISTER( imm );
    char *buf;

    unsigned long int oldPC = pc;


    reg = GETBITS( pc, 0x0E00, 9 );
    opmode = GETBITS( pc, 0x01C0, 6 );
    mode = GETBITS( pc, 0x0038, 3 );
    eaReg = GETBITS( pc, 0x0007, 0 );

    // dn,an ( an - dn )
    if( mode == 0 )
    {
        if( opmode == 3 )
        {
            result.w = a[ reg ].w - d[ eaReg ].w;
            SUB_SETC( d[ eaReg ].w, a[ reg ].w, result.w, MSB16 );
            SUB_SETV( d[ eaReg ].w, a[ reg ].w, result.w, MSB16 );
            a[ reg ].w = result.w;
            SETNZW( a[ reg ].w );
        }
        if( opmode == 7 )
        {
            result.l = a[ reg ].l - d[ eaReg ].l;
            SUB_SETC( d[ eaReg ].l, a[ reg ].l, result.l, MSB32 );
            SUB_SETV( d[ eaReg ].l, a[ reg ].l, result.l, MSB32 );
            a[ reg ].l = result.l;
            SETNZL( a[ reg ].l );
        }
        pc += 2;
    }
    // an, an
        if( mode == 1 )
    {
        if( opmode == 3 )
        {
            result.w = a[ reg ].w - a[ eaReg ].w;
            SUB_SETC( a[ reg ].w, d[ eaReg ].w, result.w, MSB16 );
            SUB_SETV( a[ reg ].w, d[ eaReg ].w, result.w, MSB16 );
            a[ reg ].w = result.w;
            SETNZW( a[ reg ].w );
        }
        if( opmode == 7 )
```

```
    {
        result.l = a[ reg ].l - a[ eaReg ].l;
        SUB_SETC( a[ reg ].l, d[ eaReg ].l, result.l, MSB32 );
        SUB_SETV( a[ reg ].l, d[ eaReg ].l, result.l, MSB32 );
        a[ reg ].l = result.l;
        SETNZL( a[ reg ].l );
    }
    pc += 2;
}
// ( an ), an
if( mode == 2 )
{
    if( opmode == 3 )
    {
        result.w = (unsigned)a[ reg ].w - memory->GetWord( a[ eaReg ].l );
        SUB_SETC( memory->GetWord( a[ eaReg ].l ), (unsigned)a[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( a[ eaReg ].l ), (unsigned)a[ reg ].w, result.w, MSB16 );
        a[ reg ].w = result.w;
        SETNZW( a[ reg ].w );
    }
    if( opmode == 7 )
    {
        result.l = a[ reg ].l - memory->GetLongword( a[ eaReg ].l);
        SUB_SETC( memory->GetLongword( a[ eaReg ].l ), (unsigned)a[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( a[ eaReg ].l ), (unsigned)a[ reg ].l, result.l, MSB32 );
        a[ reg ].l = result.l;
        SETNZL( a[ reg ].l );
    }
    pc += 2;
}
// ( an )+, an
if( mode == 3 )
{
    if( opmode == 3 )
    {
        result.w = (unsigned)a[ reg ].w - memory->GetWord( a[ eaReg ].l );
        SUB_SETC( memory->GetWord( a[ eaReg ].l ), (unsigned)a[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( a[ eaReg ].l ), (unsigned)a[ reg ].w, result.w, MSB16 );
        a[ reg ].w = result.w;
        SETNZW( a[ reg ].w );
        a[ eaReg ].l += 2;

    }
    if( opmode == 7 )
    {
        result.l = a[ reg ].l - memory->GetLongword( a[ eaReg ].l);
        SUB_SETC( memory->GetLongword( a[ eaReg ].l ), (unsigned)a[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( a[ eaReg ].l ), (unsigned)a[ reg ].l, result.l, MSB32 );
        a[ reg ].l = result.l;
        SETNZL( a[ reg ].l );
        a[ eaReg ].l += 4;
    }
```

```
    pc += 2;
}
// -( an ), an
if( mode == 4 )
{
    if( opmode == 3 )
    {
        a[ eaReg ].l -= 2;
        result.w = (unsigned)a[ reg ].w - memory->GetWord( a[ eaReg ].l );
        SUB_SETC( memory->GetWord( a[ eaReg ].l ), (unsigned)a[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( a[ eaReg ].l ), (unsigned)a[ reg ].w, result.w, MSB16 );
        a[ reg ].w = result.w;
        SETNZW( a[ reg ].w );

    }
    if( opmode == 7 )
    {
        a[ eaReg ].l -= 4;
        result.l = a[ reg ].l - memory->GetLongword( a[ eaReg ].l);
        SUB_SETC( memory->GetLongword( a[ eaReg ].l ), (unsigned)a[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( a[ eaReg ].l ), (unsigned)a[ reg ].l, result.l, MSB32 );
        a[ reg ].l = result.l;
        SETNZL( a[ reg ].l );

    }
    pc += 2;
}
// d16(an), an
if( mode == 5 )
{
    addr = a[ eaReg ].l + (signed short int)memory->GetWord( pc + 2 );
    if( opmode == 3 )
    {
        result.w = a[ reg ].w - memory->GetWord( addr );
        SUB_SETC( memory->GetWord( addr ), a[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( addr ), a[ reg ].w, result.w, MSB16 );
        a[ reg ].w = result.w;
        SETNZW( a[ reg ].w );
    }
    if( opmode == 7 )
    {
        result.l = a[ reg ].l - memory->GetLongword( addr );
        SUB_SETC( memory->GetLongword( addr ), a[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( addr ), a[ reg ].l, result.l, MSB32 );
        a[ reg ].l = result.l;
        SETNZL( a[ reg ].l );
    }
    pc += 4;
}
// d8(an), an
if( mode == 6 )
{
```

```
    addr = a[ eaReg ].l + (signed char)memory->GetByte( pc + 3 )
        + d[ GETBITS( pc + 2, 0xf000, 12 )].l;
    if ( opmode == 3 )
    {
        result.w = a[ reg ].w - memory->GetWord( addr );
        SUB_SETC( memory->GetWord( addr ), a[ reg ].w, result.w, MSB16 );
        SUB_SETV( memory->GetWord( addr ), a[ reg ].w, result.w, MSB16 );
        a[ reg ].w = result.w;
        SETNZW( a[ reg ].w );
    }
    if ( opmode == 7 )
    {
        result.l = a[ reg ].l - memory->GetLongword( addr );
        SUB_SETC( memory->GetLongword( addr ), a[ reg ].l, result.l, MSB32 );
        SUB_SETV( memory->GetLongword( addr ), a[ reg ].l, result.l, MSB32 );
        a[ reg ].l = result.l;
        SETNZL( a[ reg ].l );
    }
    pc += 4;
}

// #<data>,an
if ( ( mode == 7 )&& ( eaReg == 4 ))
{
    if ( opmode == 3 )
    {
        imm.w = (signed short int)memory->GetWord( pc + 2 );
        result.w = a[ reg ].w - imm.w;
        SETNZW( result.w );
        SUB_SETC( imm.w, a[ reg ].w, result.w, MSB16 );
        SUB_SETV( imm.w, a[ reg ].w, result.w, MSB16 );
        a[ reg ].w = result.w;
        pc += 4;
    }
    if ( opmode == 7 )
    {
        imm.l = (signed long int)memory->GetLongword( pc + 2 );
        result.l = a[ reg ].l - imm.l;
        SETNZL( result.l );
        SUB_SETC( imm.l, a[ reg ].l, result.l, MSB32 );
        SUB_SETV( imm.l, a[ reg ].l, result.l, MSB32 );
        a[ reg ].l = result.l;
        pc += 6;
    }
    return;
}

//(XXX).W, an
if ( mode == 7 )
{
    if ( eaReg == 0 )
    {
```

```
            addr = memory->GetWord( pc + 2 );
            pc += 4;
        }
        else
        {
            //(XXX).L, an
            addr = memory->GetLongword( pc + 2 );
            pc += 6;
        }
        if ( opmode == 3 )
        {
            result.w = a[ reg ].w
                - memory->GetWord( addr );
            SETNZW( result.w );
            SUB_SETC( memory->GetWord( addr ), a[ reg ].w, result.w, MSB16 );
            SUB_SETV( memory->GetWord( addr ), a[ reg ].w, result.w, MSB16 );
            a[ reg ].w = result.w;
        }
        if ( opmode == 7 )
        {
            result.l = a[ reg ].l
                - memory->GetLongword( addr );
            SETNZL( result.l );
            SUB_SETC( memory->GetLongword( addr ), a[ reg ].l, result.l, MSB32 );
            SUB_SETV( memory->GetLongword( addr ), a[ reg ].l, result.l, MSB32 );
            a[ reg ].l = result.l;
        }
    }

    if ( reg == 7 )
    {
        buf = new char[ 128 ];
        sprintf( buf, "WARNING at address $%lX: You probably shouldn't edit the SSP (a7) directly.", oldPC
            );
        helpStack.push( buf );
    }
}

void M68008::subi()
{
    int size, mode, reg, displ;
    DATA_REGISTER( result );
    DATA_REGISTER( imm );


    size = GETBITS( pc, 0x00C0, 6 );
    mode = GETBITS( pc, 0x0038, 3 );
    reg = GETBITS( pc, 0x0007, 0 );

    // #data,dn
    if ( mode == 0 )
    {
```

```
    if( size == 0 )
    {
        imm.b = memory->GetByte( pc + 3 );
        result.b = d[ reg ].b - imm.b;
        SETNZB( result.b );
        SUB_SETC( imm.b, d[ reg ].b, result.b, MSB8 );
        SUB_SETV( imm.b, d[ reg ].b, result.b, MSB8 );
        d[ reg ].b = result.b;
        pc += 4;
    }
    if( size == 1 )
    {
        imm.w = memory->GetWord( pc + 2 );
        result.w = d[ reg ].w - imm.w;
        SETNZW( result.w );
        SUB_SETC( imm.w, d[ reg ].w, result.w, MSB16 );
        SUB_SETV( imm.w, d[ reg ].w, result.w, MSB16 );
        d[ reg ].w = result.w;
        pc += 4;
    }
    if( size == 2 )
    {
        imm.l = memory->GetLongword( pc + 2 );
        result.l = d[ reg ].l - imm.l;
        SETNZL( result.l );
        SUB_SETC( imm.l, d[ reg ].l, result.l, MSB32 );
        SUB_SETV( imm.l, d[ reg ].l, result.l, MSB32 );
        d[ reg ].l = result.l;
        pc += 6;
    }
}

// #data,(an)
if( mode == 2 )
{
    if( size == 0 )
    {
        imm.b = memory->GetByte( pc + 3 );
        result.b = memory->GetByte( a[ reg ].l )- imm.b;
        SETNZB( result.b );
        SUB_SETC( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        SUB_SETV( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        memory->SetByte( result.b, a[ reg ].l );
        pc += 4;
    }
    if( size == 1 )
    {
        imm.w = memory->GetWord( pc + 2 );
        result.w = memory->GetWord( a[ reg ].l )- imm.w;
        SETNZW( result.w );
        SUB_SETC( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        SUB_SETV( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
```

```
        memory->SetWord( result.w, a[ reg ].l );
        pc += 4;
    }
    if( size == 2 )
    {
        imm.l = memory->GetLongword( pc + 2 );
        result.l = memory->GetLongword( a[ reg ].l ) - imm.l;
        SETNZL( result.l );
        SUB_SETC( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        SUB_SETV( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        memory->SetLongword( result.l, a[ reg ].l );
        pc += 6;
    }
}

// #data, (an)+
if( mode == 3 )
{
    if( size == 0 )
    {
        imm.b = memory->GetByte( pc + 3 );
        result.b = memory->GetByte( a[ reg ].l ) - imm.b;
        SETNZB( result.b );
        SUB_SETC( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        SUB_SETV( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        memory->SetByte( result.b, a[ reg ].l );
        pc += 4;
        a[ reg ].l += 1;
    }
    if( size == 1 )
    {
        imm.w = memory->GetWord( pc + 2 );
        result.w = memory->GetWord( a[ reg ].l ) - imm.w;
        SETNZW( result.w );
        SUB_SETC( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        SUB_SETV( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        memory->SetWord( result.w, a[ reg ].l );
        pc += 4;
        a[ reg ].l += 2;
    }
    if( size == 2 )
    {
        imm.l = memory->GetLongword( pc + 2 );
        result.l = memory->GetLongword( a[ reg ].l ) - imm.l;
        SETNZL( result.l );
        SUB_SETC( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        SUB_SETV( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        memory->SetLongword( result.l, a[ reg ].l );
        pc += 6;
        a[ reg ].l += 4;
    }
}
```

```
// #data, -(an)
if( mode == 4 )
{
    if( size == 0 )
    {
        a[ reg ].l -= 1;
        imm.b = memory->GetByte( pc + 3 );
        result.b = memory->GetByte( a[ reg ].l )- imm.b;
        SETNZB( result.b );
        SUB_SETC( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        SUB_SETV( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        memory->SetByte( result.b, a[ reg ].l );
        pc += 4;
    }
    if( size == 1 )
    {
        a[ reg ].l -= 2;
        imm.w = memory->GetWord( pc + 2 );
        result.w = memory->GetWord( a[ reg ].l )- imm.w;
        SETNZW( result.w );
        SUB_SETC( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        SUB_SETV( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        memory->SetWord( result.w, a[ reg ].l );
        pc += 4;
    }
    if( size == 2 )
    {
        a[ reg ].l -= 4;
        imm.l = memory->GetLongword( pc + 2 );
        result.l = memory->GetLongword( a[ reg ].l )- imm.l;
        SETNZL( result.l );
        SUB_SETC( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        SUB_SETV( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        memory->SetLongword( result.l, a[ reg ].l );
        pc += 6;
    }
}

// #data, d16(an)
if( mode == 5 )
{
    if( size == 0 )
    {
        displ = (signed short int)memory->GetWord( pc + 4 );
        imm.b = memory->GetByte( pc + 3 );
        result.b = memory->GetByte( a[ reg ].l + displ )- imm.b;
        SETNZB( result.b );
        SUB_SETC( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        SUB_SETV( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        memory->SetByte( result.b, a[ reg ].l + displ );
        pc += 6;
```

```
    }
    if( size == 1 )
    {
        displ = (signed short int)memory->GetWord( pc + 4 );
        imm.w = memory->GetWord( pc + 2 );
        result.w = memory->GetWord( a[ reg ].l + displ ) - imm.w;
        SETNZW( result.w );
        SUB_SETC( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        SUB_SETV( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        memory->SetWord( result.w, a[ reg ].l + displ );
        pc += 6;
    }
    if( size == 2 )
    {
        displ = (signed short int)memory->GetWord( pc + 6 );
        imm.l = memory->GetLongword( pc + 2 );
        result.l = memory->GetLongword( a[ reg ].l + displ ) - imm.l;
        SETNZL( result.l );
        SUB_SETC( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        SUB_SETV( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        memory->SetLongword( result.l, a[ reg ].l + displ );
        pc += 8;
    }
}

// #data, d8(an,xn)
if( mode == 6 )
{
    if( size == 0 )
    {
        displ = (signed char)memory->GetByte( pc + 5 )
            + d[ GETBITS( pc + 4, 0xF000, 12 )].l;
        imm.b = memory->GetByte( pc + 3 );
        result.b = memory->GetByte( a[ reg ].l + displ ) - imm.b;
        SETNZB( result.b );
        SUB_SETC( imm.b, memory->GetByte( a[ reg ].l + displ ), result.b, MSB8 );
        SUB_SETV( imm.b, memory->GetByte( a[ reg ].l + displ ), result.b, MSB8 );
        memory->SetByte( result.b, a[ reg ].l + displ );
        pc += 6;
    }
    if( size == 1 )
    {
        displ = (signed char)memory->GetByte( pc + 5 )
            + d[ GETBITS( pc + 4, 0xF000, 12 )].l;
        imm.w = memory->GetWord( pc + 2 );
        result.w = memory->GetWord( a[ reg ].l + displ ) - imm.w;
        SETNZW( result.w );
        SUB_SETC( imm.w, memory->GetWord( a[ reg ].l + displ ), result.w, MSB16 );
        SUB_SETV( imm.w, memory->GetWord( a[ reg ].l + displ ), result.w, MSB16 );
        memory->SetWord( result.w, a[ reg ].l + displ );
        pc += 6;
    }
```

```
    if( size == 2 )
    {
        displ = (signed char)memory->GetByte( pc + 7 )
            + d[ GETBITS( pc + 4, 0xF000, 12 )].l;
        imm.l = memory->GetLongword( pc + 2 );
        result.l = memory->GetLongword( a[ reg ].l + displ )- imm.l;
        SETNZL( result.l );
        SUB_SETC( imm.l, memory->GetLongword( a[ reg ].l + displ ), result.l, MSB32 );
        SUB_SETV( imm.l, memory->GetLongword( a[ reg ].l + displ ), result.l, MSB32 );
        memory->SetLongword( result.l, a[ reg ].l + displ );
        pc += 8;
    }
}

if( mode == 7 )
{
    // #data, (xxx).w
    if( reg == 0 )
    {
        if( size == 0 )
        {
            displ = memory->GetWord( pc + 4 );
            imm.b = memory->GetByte( pc + 3 );
            result.b = memory->GetByte( displ )- imm.b;
            SETNZB( result.b );
            SUB_SETC( imm.b, memory->GetByte( displ ), result.b, MSB8 );
            SUB_SETV( imm.b, memory->GetByte( displ ), result.b, MSB8 );
            memory->SetByte( result.b, displ );
            pc += 6;
        }
        if( size == 1 )
        {
            displ = memory->GetWord( pc + 4 );
            imm.w = memory->GetWord( pc + 2 );
            result.w = memory->GetWord( displ )- imm.w;
            SETNZW( result.w );
            SUB_SETC( imm.w, memory->GetWord( displ ), result.w, MSB16 );
            SUB_SETV( imm.w, memory->GetWord( displ ), result.w, MSB16 );
            memory->SetWord( result.w, displ );
            pc += 6;
        }
        if( size == 2 )
        {
            displ = memory->GetWord( pc + 6 );
            imm.l = memory->GetLongword( pc + 2 );
            result.l = memory->GetLongword( displ )- imm.l;
            SETNZL( result.l );
            SUB_SETC( imm.l, memory->GetLongword( displ ), result.l, MSB32 );
            SUB_SETV( imm.l, memory->GetLongword( displ ), result.l, MSB32 );
            memory->SetLongword( result.l, displ );
            pc += 8;
        }
```

```
        }

        // #data, (xxx).l
        if( reg == 1 )
        {
            if( size == 0 )
            {
                displ = memory->GetLongword( pc + 4 );
                imm.b = memory->GetByte( pc + 3 );
                result.b = memory->GetByte( displ )- imm.b;
                SETNZB( result.b );
                SUB_SETC( imm.b, memory->GetByte( displ ), result.b, MSB8 );
                SUB_SETV( imm.b, memory->GetByte( displ ), result.b, MSB8 );
                memory->SetByte( result.b, displ );
                pc += 8;
            }
            if( size == 1 )
            {
                displ = memory->GetLongword( pc + 4 );
                imm.w = memory->GetWord( pc + 2 );
                result.w = memory->GetWord( displ )- imm.w;
                SETNZW( result.w );
                SUB_SETC( imm.w, memory->GetWord( displ ), result.w, MSB16 );
                SUB_SETV( imm.w, memory->GetWord( displ ), result.w, MSB16 );
                memory->SetWord( result.w, displ );
                pc += 8;
            }
            if( size == 2 )
            {
                displ = memory->GetLongword( pc + 6 );
                imm.l = memory->GetLongword( pc + 2 );
                result.l = memory->GetLongword( displ )- imm.l;
                SETNZL( result.l );
                SUB_SETC( imm.l, memory->GetLongword( displ ), result.l, MSB32 );
                SUB_SETV( imm.l, memory->GetLongword( displ ), result.l, MSB32 );
                memory->SetLongword( result.l, displ );
                pc += 10;
            }
        }
    }
}

void M68008::subq()
{
    int size, mode, reg,displ;
    DATA_REGISTER( result );
    DATA_REGISTER( imm );

    imm.l = GETBITS( pc, 0x0E00, 9 );
    size = GETBITS( pc, 0x00C0, 6 );
    mode = GETBITS( pc, 0x0038, 3 );
    reg = GETBITS( pc, 0x0007, 0 );
```

```
// #data,dn
if ( mode == 0 )
{
    if ( size == 0 )
    {
        result.b = d[ reg ].b - imm.b;
        SETNZB( result.b );
        SUB_SETC( imm.b, d[ reg ].b, result.b, MSB8 );
        SUB_SETV( imm.b, d[ reg ].b, result.b, MSB8 );
        d[ reg ].b = result.b;
        pc += 2;
    }
    if ( size == 1 )
    {
        result.w = d[ reg ].w - imm.w;
        SETNZW( result.w );
        SUB_SETC( imm.w, d[ reg ].w, result.w, MSB16 );
        SUB_SETV( imm.w, d[ reg ].w, result.w, MSB16 );
        d[ reg ].w = result.w;
        pc += 2;
    }
    if ( size == 2 )
    {
        result.l = d[ reg ].l - imm.l;
        SETNZL( result.l );
        SUB_SETC( imm.l, d[ reg ].l, result.l, MSB32 );
        SUB_SETV( imm.l, d[ reg ].l, result.l, MSB32 );
        d[ reg ].l = result.l;
        pc += 2;
    }
}

// #data,an
if ( mode == 1 )
{
    if ( size == 1 )
    {
        result.w = a[ reg ].w - imm.w;
        SETNZW( result.w );
        SUB_SETC( imm.w, a[ reg ].w, result.w, MSB16 );
        SUB_SETV( imm.w, a[ reg ].w, result.w, MSB16 );
        a[ reg ].w = result.w;
        pc += 2;
    }
    if ( size == 2 )
    {
        result.l = a[ reg ].l - imm.l;
        SETNZL( result.l );
        SUB_SETC( imm.l, a[ reg ].l, result.l, MSB32 );
        SUB_SETV( imm.l, a[ reg ].l, result.l, MSB32 );
        a[ reg ].l = result.l;
```

```
        pc += 2;
    }
}

// #data,(an)
if( mode == 2 )
{
    if( size == 0 )
    {
        result.b = memory->GetByte( a[ reg ].l )- imm.b;
        SETNZB( result.b );
        SUB_SETC( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        SUB_SETV( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        memory->SetByte( result.b, a[ reg ].l );
        pc += 2;
    }
    if( size == 1 )
    {
        result.w = memory->GetWord( a[ reg ].l )- imm.w;
        SETNZW( result.w );
        SUB_SETC( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        SUB_SETV( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        memory->SetWord( result.w, a[ reg ].l );
        pc += 2;
    }
    if( size == 2 )
    {
        result.l = memory->GetLongword( a[ reg ].l )- imm.l;
        SETNZL( result.l );
        SUB_SETC( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        SUB_SETV( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        memory->SetLongword( result.l, a[ reg ].l );
        pc += 2;
    }
}

// #data, (an)+
if( mode == 3 )
{
    if( size == 0 )
    {
        result.b = memory->GetByte( a[ reg ].l )- imm.b;
        SETNZB( result.b );
        SUB_SETC( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        SUB_SETV( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        memory->SetByte( result.b, a[ reg ].l );
        pc += 2;
        a[ reg ].l += 1;
    }
    if( size == 1 )
    {
        result.w = memory->GetWord( a[ reg ].l )- imm.w;
```

```
        SETNZW( result.w );
        SUB_SETC( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        SUB_SETV( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        memory->SetWord( result.w, a[ reg ].l );
        pc += 2;
        a[ reg ].l += 2;
    }
    if( size == 2 )
    {
        result.l = memory->GetLongword( a[ reg ].l ) - imm.l;
        SETNZL( result.l );
        SUB_SETC( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        SUB_SETV( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        memory->SetLongword( result.l, a[ reg ].l );
        pc += 2;
        a[ reg ].l += 4;
    }
}

// #data, -(an)
if( mode == 4 )
{
    if( size == 0 )
    {
        a[ reg ].l -= 1;
        result.b = memory->GetByte( a[ reg ].l ) - imm.b;
        SETNZB( result.b );
        SUB_SETC( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        SUB_SETV( imm.b, memory->GetByte( a[ reg ].l ), result.b, MSB8 );
        memory->SetByte( result.b, a[ reg ].l );
        pc += 2;
    }
    if( size == 1 )
    {
        a[ reg ].l -= 2;
        result.w = memory->GetWord( a[ reg ].l ) - imm.w;
        SETNZW( result.w );
        SUB_SETC( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        SUB_SETV( imm.w, memory->GetWord( a[ reg ].l ), result.w, MSB16 );
        memory->SetWord( result.w, a[ reg ].l );
        pc += 2;
    }
    if( size == 2 )
    {
        a[ reg ].l -= 4;
        result.l = memory->GetLongword( a[ reg ].l ) - imm.l;
        SETNZL( result.l );
        SUB_SETC( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        SUB_SETV( imm.l, memory->GetLongword( a[ reg ].l ), result.l, MSB32 );
        memory->SetLongword( result.l, a[ reg ].l );
        pc += 2;
    }
```

```
}

// #data, d16(an)
if( mode == 5 )
{
    displ = (signed short int)memory->GetWord( pc + 2 );
    if( size == 0 )
    {
        result.b = memory->GetByte( a[ reg ].l + displ )- imm.b;
        SETNZB( result.b );
        SUB_SETC( imm.b, memory->GetByte( a[ reg ].l + displ ), result.b, MSB8 );
        SUB_SETV( imm.b, memory->GetByte( a[ reg ].l + displ ), result.b, MSB8 );
        memory->SetByte( result.b, a[ reg ].l + displ );
        pc += 4;
    }
    if( size == 1 )
    {
        result.w = memory->GetWord( a[ reg ].l + displ )- imm.w;
        SETNZW( result.w );
        SUB_SETC( imm.w, memory->GetWord( a[ reg ].l + displ ), result.w, MSB16 );
        SUB_SETV( imm.w, memory->GetWord( a[ reg ].l + displ ), result.w, MSB16 );
        memory->SetWord( result.w, a[ reg ].l + displ );
        pc += 4;
    }
    if( size == 2 )
    {
        result.l = memory->GetLongword( a[ reg ].l + displ )- imm.l;
        SETNZL( result.l );
        SUB_SETC( imm.l, memory->GetLongword( a[ reg ].l + displ ), result.l, MSB32 );
        SUB_SETV( imm.l, memory->GetLongword( a[ reg ].l + displ ), result.l, MSB32 );
        memory->SetLongword( result.l, a[ reg ].l + displ );
        pc += 4;
    }
}

// #data, d8(an,xn)
if( mode == 6 )
{
    displ = (signed char)memory->GetByte( pc + 3 )
        + d[ GETBITS( pc + 2, 0xF000, 12 )].l;
    if( size == 0 )
    {
        result.b = memory->GetByte( a[ reg ].l + displ )- imm.b;
        SETNZB( result.b );
        SUB_SETC( imm.b, memory->GetByte( a[ reg ].l + displ ), result.b, MSB8 );
        SUB_SETV( imm.b, memory->GetByte( a[ reg ].l + displ ), result.b, MSB8 );
        memory->SetByte( result.b, a[ reg ].l + displ );
        pc += 4;
    }
    if( size == 1 )
    {
        result.w = memory->GetWord( a[ reg ].l + displ )- imm.w;
```

```
        SETNZW( result.w );
        SUB_SETC( imm.w, memory->GetWord( a[ reg ].l + displ ), result.w, MSB16 );
        SUB_SETV( imm.w, memory->GetWord( a[ reg ].l + displ ), result.w, MSB16 );
        memory->SetWord( result.w, a[ reg ].l + displ );
        pc += 4;
    }
    if( size == 2 )
    {
        result.l = memory->GetLongword( a[ reg ].l + displ )- imm.l;
        SETNZL( result.l );
        SUB_SETC( imm.l, memory->GetLongword( a[ reg ].l + displ ), result.l, MSB32 );
        SUB_SETV( imm.l, memory->GetLongword( a[ reg ].l + displ ), result.l, MSB32 );
        memory->SetLongword( result.l, a[ reg ].l + displ );
        pc += 4;
    }
}

if( mode == 7 )
{
    // #data, (xxx).w
    if( reg == 0 )
    {
        displ = memory->GetWord( pc + 2 );
        if( size == 0 )
        {
            result.b = memory->GetByte( displ )- imm.b;
            SETNZB( result.b );
            SUB_SETC( imm.b, memory->GetByte( displ ), result.b, MSB8 );
            SUB_SETV( imm.b, memory->GetByte( displ ), result.b, MSB8 );
            memory->SetByte( result.b, displ );
        }
        if( size == 1 )
        {
            result.w = memory->GetWord( displ )- imm.w;
            SETNZW( result.w );
            SUB_SETC( imm.w, memory->GetWord( displ ), result.w, MSB16 );
            SUB_SETV( imm.w, memory->GetWord( displ ), result.w, MSB16 );
            memory->SetWord( result.w, displ );
        }
        if( size == 2 )
        {
            result.l = memory->GetLongword( displ )- imm.l;
            SETNZL( result.l );
            SUB_SETC( imm.l, memory->GetLongword( displ ), result.l, MSB32 );
            SUB_SETV( imm.l, memory->GetLongword( displ ), result.l, MSB32 );
            memory->SetLongword( result.l, displ );
        }
        pc += 4;
    }

    // #data, (xxx).l
    if( reg == 1 )
```

```
        {
            displ = memory->GetLongword( pc + 2 );
            if( size == 0 )
            {
                result.b = memory->GetByte( displ )- imm.b;
                SETNZB( result.b );
                SUB_SETC( imm.b, memory->GetByte( displ ), result.b, MSB8 );
                SUB_SETV( imm.b, memory->GetByte( displ ), result.b, MSB8 );
                memory->SetByte( result.b, displ );
            }
            if( size == 1 )
            {
                result.w = memory->GetWord( displ )- imm.w;
                SETNZW( result.w );
                SUB_SETC( imm.w, memory->GetWord( displ ), result.w, MSB16 );
                SUB_SETV( imm.w, memory->GetWord( displ ), result.w, MSB16 );
                memory->SetWord( result.w, displ );
            }
            if( size == 2 )
            {
                result.l = memory->GetLongword( displ )- imm.l;
                SETNZL( result.l );
                SUB_SETC( imm.l, memory->GetLongword( displ ), result.l, MSB32 );
                SUB_SETV( imm.l, memory->GetLongword( displ ), result.l, MSB32 );
                memory->SetLongword( result.l, displ );
            }
            pc += 6;
        }
    }
}

void M68008::subx()
{
    int regY, size, rm, regX;
    DATA_REGISTER( result );


    regY = GETBITS( pc, 0x0E00, 9 );
    size = GETBITS( pc, 0x00C0, 6 );
    rm = GETBITS( pc, 0x0008, 3 );
    regX = GETBITS( pc, 0x0007, 0 );

    if( size == 3 ) // Instruction is in fact SUBA
    {
        suba();
        return;
    }

    // subx dx,dy
    if( rm == 0 )
    {
        if( size == 0 )
```

```
    {
        result.b = d[ regY ].b - d[ regX ].b - ccr.x;
        SETNZB( result.b );
        SUB_SETC( d[ regX ].b, d[ regY ].b, result.b, MSB8 );
        SUB_SETV( d[ regX ].b, d[ regY ].b, result.b, MSB8 );
        d[ regY ].b = result.b;
    }
    if( size == 1 )
    {
        result.w = d[ regY ].w - d[ regX ].w - ccr.x;
        SETNZW( result.w );
        SUB_SETC( d[ regX ].w, d[ regY ].w, result.w, MSB16 );
        SUB_SETV( d[ regX ].w, d[ regY ].w, result.w, MSB16 );
        d[ regY ].w = result.w;
    }
    if( size == 2 )
    {
        result.l = d[ regY ].l - d[ regX ].l - ccr.x;
        SETNZL( result.l );
        SUB_SETC( d[ regX ].l, d[ regY ].l, result.l, MSB32 );
        SUB_SETV( d[ regX ].l, d[ regY ].l, result.l, MSB32 );
        d[ regY ].l = result.l;
    }
    pc += 2;
}

// subx (-ax),-(ay)
if( rm == 1 )
{
    if( size == 0 )
    {
        a[ regX ].l -= 1;
        a[ regY ].l -= 1;
        result.b = memory->GetByte( a[ regY ].l )- memory->GetByte( a[ regX ].l )- ccr.x;
        SETNZB( result.b );
        SUB_SETC( memory->GetByte( a[ regX ].l ), memory->GetByte( a[ regY ].l ), result.b, MSB8 );
        SUB_SETV( memory->GetByte( a[ regX ].l ), memory->GetByte( a[ regY ].l ), result.b, MSB8 );
        memory->SetByte( result.b, a[ regY ].l );
    }
    if( size == 1 )
    {
        a[ regX ].l -= 2;
        a[ regY ].l -= 2;
        result.w = memory->GetWord( a[ regY ].l )- memory->GetWord( a[ regX ].l )- ccr.x;
        SETNZW( result.w );
        SUB_SETC( memory->GetWord( a[ regX ].l ), memory->GetWord( a[ regY ].l ), result.w, MSB16 );
        SUB_SETV( memory->GetWord( a[ regX ].l ), memory->GetWord( a[ regY ].l ), result.w, MSB16 );
        memory->SetWord( result.w, a[ regY ].l );
    }
    if( size == 2 )
    {
        a[ regX ].l -= 4;
```

```
            a[ regY ].l -= 4;
            result.l = memory->GetLongword( a[ regY ].l )- memory->GetLongword( a[ regX ].l )- ccr.x;
            SETNZL( result.l );
            SUB_SETC( memory->GetLongword( a[ regX ].l ), memory->GetLongword( a[ regY ].l ), result.l,
                MSB32 );
            SUB_SETV( memory->GetLongword( a[ regX ].l ), memory->GetLongword( a[ regY ].l ), result.l,
                MSB32 );
            memory->SetLongword( result.l, a[ regY ].l );
        }
        pc += 2;
    }

}

void M68008::swap()
{
    unsigned long int reg, regLow, regHigh;

    reg = GETBITS( pc, 7, 0 );

    regLow = d[ reg ].l & 0x0000FFFF;
    regHigh = ( d[ reg ].l & 0xFFFF0000 )>> 16;

    regLow ^= regHigh;
    regHigh ^= regLow;
    regLow ^= regHigh;

    d[ reg ].l = regLow | ( regHigh << 16 );
    pc += 2;
}

void M68008::tas()
{
    int mode, reg, xn;
    unsigned long displ;

    mode = GETBITS( pc, 0x0038, 3 );
    reg  = GETBITS( pc, 0x0007, 0 );

    ccr.v = ccr.c = 0;

    // dn
    if( mode == 0 )
    {
        SETNZB( d[ reg ].b );
        d[ reg ].b |= MSB8;
        pc += 2;
    }

    // (an)
    if( mode == 2 )
    {
```

```
    SETNZB( memory->GetByte( a[ reg ].l ));
    memory->SetByte( memory->GetByte( a[ reg ].l )| MSB8, a[ reg ].l );
    pc += 2;
}

// (an)+
if( mode == 3 )
{
    SETNZB( memory->GetByte( a[ reg ].l ));
    memory->SetByte( memory->GetByte( a[ reg ].l )| MSB8, a[ reg ].l );
    a[ reg ].l++;
    pc += 2;
}

// -(an)
if( mode == 4 )
{
    a[ reg ].l--;
    SETNZB( memory->GetByte( a[ reg ].l ));
    memory->SetByte( memory->GetByte( a[ reg ].l )| MSB8, a[ reg ].l );
    pc += 2;
}

// d16(an)
if( mode == 5 )
{
    displ = memory->GetWord( pc + 2 )+ a[ reg ].l;
    SETNZB( memory->GetByte( displ ));
    memory->SetByte( memory->GetByte( displ )| MSB8, displ );
    pc += 4;
}

// d8(an,xn)
if( mode == 6 )
{
    xn = GETBITS( pc + 2, 0xF000, 12 );
    displ = memory->GetByte( pc + 3 )+ a[ reg ].l + d[ xn ].l;
    SETNZB( memory->GetByte( displ ));
    memory->SetByte( memory->GetByte( displ )| MSB8, displ );
    pc += 4;
}

if( mode == 7 )
{
    // (xxx).w
    if( reg == 0 )
    {
        displ = memory->GetWord( pc + 2 );
        pc += 4;
    }

    // (xxx).l
```

```
        if( reg == 1 )
        {
            displ = memory->GetLongword( pc + 2 );
            pc += 6;
        }

        SETNZB( memory->GetByte( displ ));
        memory->SetByte( memory->GetByte( displ )| MSB8, displ );
    }
}

void M68008::trap()
{
    int vector;
    short int CCR;

    vector = GETBITS( pc, 0x000f, 0 )+ 32;
    CCR = ccr.x << 4 | ccr.n << 3 | ccr.z << 2 | ccr.v << 1 | ccr.c;

    a [ 7 ]. l -= 4;
    memory->SetLongword( pc + 2, a[ 7 ].l );
    a [ 7 ]. l -= 2;
    memory->SetWord( CCR, a[ 7 ].l );

    PUSH( "PC (by TRAP)", 4 );
    PUSH( "CCR (by TRAP)", 2 );

    pc = memory->GetLongword( vector * 4 );
}

void M68008::trapv()
{
    short int CCR;


    if( ccr.v )
    {
        CCR = ccr.x << 4 | ccr.n << 3 | ccr.z << 2 | ccr.v << 1 | ccr.c;

        a [ 7 ]. l -= 4;
        memory->SetLongword( pc + 2, a[ 7 ].l );
        a [ 7 ]. l -= 2;
        memory->SetWord( CCR, a[ 7 ].l );

        PUSH( "PC (by TRAPV)", 4 );
        PUSH( "CCR (by TRAPV)", 2 );

        pc = memory->GetLongword( 7 * 4 );
    }
    else
        pc += 2;
}
```

```
void M68008::tst()
{
    int  size , mode, reg, xn;
    unsigned long displ;

    size = GETBITS( pc, 0x00C0, 6 );
    mode = GETBITS( pc, 0x0038, 3 );
    reg  = GETBITS( pc, 0x0007, 0 );

    ccr.c = ccr.v = 0;

    // dn
    if( mode == 0 )
    {
        // byte
        if( size == 0 )
        {
            SETNZB( d[ reg ].b );
        }

        // word
        if( size == 1 )
        {
            SETNZW( d[ reg ].w );
        }

        // long
        if( size == 2 )
        {
            SETNZL( d[ reg ].l );
        }

        pc += 2;
    }

    // (an)
    if( mode == 2 )
    {
        // byte
        if( size == 0 )
        {
            SETNZB( memory->GetByte( a[ reg ].l ));
        }

        // word
        if( size == 1 )
        {
            SETNZW( memory->GetWord( a[ reg ].l ));
        }

        // long
```

```
    if( size == 2 )
    {
        SETNZL( memory->GetLongword( a[ reg ].l ));
    }

    pc += 2;
}

// (an)+
if( mode == 3 )
{
    // byte
    if( size == 0 )
    {
        SETNZB( memory->GetByte( a[ reg ].l ));
        a[ reg ].l += 1;
    }

    // word
    if( size == 1 )
    {
        SETNZW( memory->GetWord( a[ reg ].l ));
        a[ reg ].l += 2;
    }

    // long
    if( size == 2 )
    {
        SETNZL( memory->GetLongword( a[ reg ].l ));
        a[ reg ].l += 4;
    }

    pc += 2;
}

// -(an)
if( mode == 4 )
{
    // byte
    if( size == 0 )
    {
        a[ reg ].l -= 1;
        SETNZB( memory->GetByte( a[ reg ].l ));
    }

    // word
    if( size == 1 )
    {
        a[ reg ].l -= 2;
        SETNZW( memory->GetWord( a[ reg ].l ));
    }
```

```
    // long
    if ( size == 2 )
    {
        a[ reg ].l -= 4;
        SETNZL( memory->GetLongword( a[ reg ].l ));
    }

    pc += 2;
}

// d16(an)
if ( mode == 5 )
{
    displ = memory->GetWord( pc + 2 )+ a[ reg ].l;

    // byte
    if ( size == 0 )
    {
        SETNZB( memory->GetByte( displ ));
    }

    // word
    if ( size == 1 )
    {
        SETNZW( memory->GetWord( displ ));
    }

    // long
    if ( size == 2 )
    {
        SETNZL( memory->GetLongword( displ ));
    }

    pc += 4;
}

// d8(an,dn)
if ( mode == 6 )
{
    xn = GETBITS( pc + 2, 0xF000, 12 );
    displ = memory->GetByte( pc + 3 )+ a[ reg ].l + d[ xn ].l;

    // byte
    if ( size == 0 )
    {
        SETNZB( memory->GetByte( displ ));
    }

    // word
    if ( size == 1 )
    {
        SETNZW( memory->GetWord( displ ));
```

```
    }

    // long
    if( size == 2 )
    {
        SETNZL( memory->GetLongword( displ ));
    }

    pc += 4;
}

if( mode == 7 )
{
    // (xxx).w
    if( reg == 0 )
    {
        displ = memory->GetWord( pc + 2 );

        // byte
        if( size == 0 )
        {
            SETNZB( memory->GetByte( displ ));
        }

        // word
        if( size == 1 )
        {
            SETNZW( memory->GetWord( displ ));
        }

        // long
        if( size == 2 )
        {
            SETNZL( memory->GetLongword( displ ));
        }

        pc += 4;
    }

    // (xxx).l
    if( reg == 1 )
    {
        displ = memory->GetLongword( pc + 2 );

        // byte
        if( size == 0 )
        {
            SETNZB( memory->GetByte( displ ));
        }

        // word
        if( size == 1 )
```

```
            {
                SETNZW( memory->GetWord( displ ));
            }

            // long
            if( size == 2 )
            {
                SETNZL( memory->GetLongword( displ ));
            }

            pc += 6;
        }
    }
}

void M68008::unlk()
{
    int reg;

    reg = GETBITS( pc, 0x0007, 0 );

    a [ 7 ]. l = a[ reg ]. l;
    a[ reg ]. l = memory->GetLongword( a[ 7 ].l );
    a [ 7 ]. l += 4;

    POP( 4 );

    pc += 2;
}
```

1.9 TInstrO-Z.cpp

The translation methods of the instructions starting with $O \cdots Z$.

```
/*
    Motorola 68008 Simulator
    (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
    Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

    This class represents the Motorola 68008 Microprocessor
    Instruction definitions O-Z String Translations
*/

#include <iostream.h>
#include <math.h>
#include <stdio.h>
#include "M68008.h"

#define MSB8 0x80
#define MSB16 0x8000
#define MSB32 0x80000000
```

```
#define GETBITS( addr, mask, shift )( ( memory->GetWord( addr )& mask )>> shift )
#define SETNZ( src )ccr.n = ( (signed char)src < 0 ); ccr.z = ( src == 0 );
#define IPOW( n, m )( (unsigned long int)pow( n, m ))
#define SUB_SETV( S, D, R, M )ccr.v = ( ( ~( S & M )& ( D & M )& ~( R & M )| ( S & M )& ~( D & M )
    & ( R & M ))& M )? 1 : 0;
#define SUB_SETC( S, D, R, M )ccr.c = ccr.x = ( ( ( S & M )& ~( D & M )| ( R & M )& ~( D & M )| ( S &
    M )& ( R & M ))& M ) ? 1 : 0;

unsigned long M68008::tS_or( unsigned long address, char *buf )
{
   int reg, opmode, mode, eaReg, xn;
   unsigned long displ;

   reg    = GETBITS( address, 0x0E00, 9 );
   opmode = GETBITS( address, 0x01C0, 6 );
   mode   = GETBITS( address, 0x0038, 3 );
   eaReg  = GETBITS( address, 0x0007, 0 );

   // dn,Dn
   if( mode == 0 )
   {
      // byte
      if( opmode == 0 )
      {
         sprintf( buf, "or.b d%X,d%X", eaReg, reg );
      }

      // word
      if( opmode == 1 )
      {
         sprintf( buf, "or.w d%X,d%X", eaReg, reg );
      }

      // long
      if( opmode == 2 )
      {
         sprintf( buf, "or.l d%X,d%X", eaReg, reg );
      }

      address += 2;
   }

   // (an),dn
   if( mode == 2 )
   {
      // byte
      if( opmode == 0 )
      {
         sprintf( buf, "or.b (a%X),d%X", eaReg, reg );
      }

      // word
```

```
    if ( opmode == 1 )
    {
        sprintf ( buf, "or.w (a%X),d%X", eaReg, reg );
    }

    // long
    if ( opmode == 2 )
    {
        sprintf ( buf, "or.l (a%X),d%X", eaReg, reg );
    }

    // byte
    if ( opmode == 4 )
    {
        sprintf ( buf, "or.b d%X,(a%X)", reg, eaReg );
    }

    // word
    if ( opmode == 5 )
    {
        sprintf ( buf, "or.w d%X,(a%X)", reg, eaReg );
    }

    // long
    if ( opmode == 6 )
    {
        sprintf ( buf, "or.l d%X,(a%X)", reg, eaReg );
    }

    address += 2;
}

// (an)+,dn
if ( mode == 3 )
{
    // byte
    if ( opmode == 0 )
    {
        sprintf ( buf, "or.b (a%X)+,d%X", eaReg, reg );
    }

    // word
    if ( opmode == 1 )
    {
        sprintf ( buf, "or.w (a%X)+,d%X", eaReg, reg );
    }

    // long
    if ( opmode == 2 )
    {
        sprintf ( buf, "or.l (a%X)+,d%X", eaReg, reg );
    }
```

```
        // byte
        if ( opmode == 4 )
        {
            sprintf ( buf, "or.b␣d%X,(a%X)+", reg, eaReg );
        }

        // word
        if ( opmode == 5 )
        {
            sprintf ( buf, "or.w␣d%X,(a%X)+", reg, eaReg );
        }

        // long
        if ( opmode == 6 )
        {
            sprintf ( buf, "or.l␣d%X,(a%X)+", reg, eaReg );
        }

        address += 2;
    }

    // -(an),dn
    if ( mode== 4 )
    {
        // byte
        if ( opmode == 0 )
        {
            sprintf ( buf, "or.b␣-(a%X),d%X", eaReg, reg );
        }

        // word
        if ( opmode == 1 )
        {
            sprintf ( buf, "or.w␣-(a%X),d%X", eaReg, reg );
        }

        // long
        if ( opmode == 2 )
        {
            sprintf ( buf, "or.l␣-(a%X),d%X", eaReg, reg );
        }

        // byte
        if ( opmode == 4 )
        {
            sprintf ( buf, "or.b␣d%X,-(a%X)", reg, eaReg );
        }

        // word
        if ( opmode == 5 )
        {
```

```
        sprintf ( buf, "or.w␣d%X,−(a%X)", reg, eaReg );
    }

    // long
    if ( opmode == 6 )
    {
        sprintf ( buf, "or.l␣d%X,−(a%X)", reg, eaReg );
    }

    address += 2;
}

// d16(an),dn
if ( mode == 5 )
{
    displ = (signed short int)memory−>GetWord( address + 2 );

    // byte
    if ( opmode == 0 )
    {
        if ( decImm )sprintf ( buf, "or.b␣%ld(a%X),d%X", displ, eaReg, reg );
        else sprintf ( buf, "or.b␣$%lX(a%X),d%X", displ, eaReg, reg );
    }

    // word
    if ( opmode == 1 )
    {
        if ( decImm )sprintf ( buf, "or.w␣%ld(a%X),d%X", displ, eaReg, reg );
        else sprintf ( buf, "or.w␣$%lX(a%X),d%X", displ, eaReg, reg );
    }

    // long
    if ( opmode == 2 )
    {
        if ( decImm )sprintf ( buf, "or.l␣%ld(a%X),d%X", displ, eaReg, reg );
        else sprintf ( buf, "or.l␣$%lX(a%X),d%X", displ, eaReg, reg );
    }

    // byte
    if ( opmode == 4 )
    {
        if ( decImm )sprintf ( buf, "or.b␣d%X,%ld(a%X)", reg, displ, eaReg );
        else sprintf ( buf, "or.b␣d%X,$%lX(a%X)", reg, displ, eaReg );
    }

    // word
    if ( opmode == 5 )
    {
        if ( decImm )sprintf ( buf, "or.w␣d%X,%ld(a%X)", reg, displ, eaReg );
        else sprintf ( buf, "or.w␣d%X,$%lX(a%X)", reg, displ, eaReg );
    }
```

```
        // long
        if ( opmode == 6 )
        {
            if ( decImm )sprintf( buf, "or.l d%X,%ld(a%X)", reg, displ, eaReg );
            else  sprintf ( buf, "or.l d%X,$%lX(a%X)", reg, displ, eaReg );
        }

        address += 4;
    }

    // d8(an,xn),dn
    if ( mode == 6 )
    {
        displ = (signed char)memory->GetByte( address + 3 );
        xn = GETBITS( address + 2, 0xF000, 12 );

        // byte
        if ( opmode == 0 )
        {
            if ( decImm )sprintf( buf, "or.b %ld(a%X,d%X),d%X", displ, eaReg, xn, reg );
            else  sprintf ( buf, "or.b $%lX(a%X,d%X),d%X", displ, eaReg, xn, reg );
        }

        // word
        if ( opmode == 1 )
        {
            if ( decImm )sprintf( buf, "or.w %ld(a%X,d%X),d%X", displ, eaReg, xn, reg );
            else  sprintf ( buf, "or.w $%lX(a%X,d%X),d%X", displ, eaReg, xn, reg );
        }

        // long
        if ( opmode == 2 )
        {
            if ( decImm )sprintf( buf, "or.l %ld(a%X,d%X),d%X", displ, eaReg, xn, reg );
            else  sprintf ( buf, "or.l $%lX(a%X,d%X),d%X", displ, eaReg, xn, reg );
        }

        // byte
        if ( opmode == 4 )
        {
            if ( decImm )sprintf( buf, "or.b d%X,%ld(a%X,d%X)", reg, displ, eaReg, xn );
            else  sprintf ( buf, "or.b d%X,$%lX(a%X,d%X)", reg, displ, eaReg, xn );
        }

        // word
        if ( opmode == 5 )
        {
            if ( decImm )sprintf( buf, "or.w d%X,%ld(a%X,d%X)", reg, displ, eaReg, xn );
            else  sprintf ( buf, "or.w d%X,$%lX(a%X,d%X)", reg, displ, eaReg, xn );
        }

        // long
```

```
    if ( opmode == 6 )
    {
        if ( decImm )sprintf ( buf, "or.l␣d%X,%ld(a%X,d%X)", reg, displ, eaReg, xn );
        else  sprintf ( buf, "or.l␣d%X,$%lX(a%X,d%X)", reg, displ, eaReg, xn );
    }

    address += 4;
}

if ( mode == 7)
{
    // (xxx).w,dn
    if ( eaReg == 0 )
    {
        displ = memory->GetWord( address + 2 );
        address += 4;
    }
    else
    {
        displ = memory->GetLongword( address + 2 );
        address += 6;
    }

    // byte
    if ( opmode == 0 )
    {
        sprintf ( buf, "or.b␣$%lX,d%X", displ, reg );
    }

    // word
    if ( opmode == 1 )
    {
        sprintf ( buf, "or.w␣$%lX,d%X", displ, reg );
    }

    // long
    if ( opmode == 2 )
    {
        sprintf ( buf, "or.l␣$%lX,d%X", displ, reg );
    }

    // byte
    if ( opmode == 4 )
    {
        sprintf ( buf, "or.b␣d%X,$%lX", reg, displ );
    }

    // word
    if ( opmode == 5 )
    {
        sprintf ( buf, "or.w␣d%X,$%lX", reg, displ );
    }
```

```
        // long
        if( opmode == 6 )
        {
            sprintf( buf, "or.l␣d%X,$%lX", reg, displ );
        }
    }

    // OMITTED
    // #data,dn (see ORI)
    // d16(PC),dn
    // d8(PC,xn)

    return address;
}

unsigned long M68008::tS_ori( unsigned long address, char *buf )
{
    int size, mode, reg, xn;
    unsigned long displ;
    DATA_REGISTER( imm );

    size = GETBITS( address, 0x00C0, 6 );
    mode = GETBITS( address, 0x0038, 3 );
    reg  = GETBITS( address, 0x0007, 0 );

    // Fetch the immediate value
    if( size == 0 )
        imm.b = memory->GetByte( address + 3 );
    if( size == 1 )
        imm.w = memory->GetWord( address + 2 );
    if( size == 2 )
        imm.l = memory->GetLongword( address + 2 );

    // #data,dn
    if( mode == 0 )
    {
        if( size == 0 )
        {
            if( decImm )sprintf( buf, "ori.b␣#%d,d%X", imm.b, reg );
            else sprintf( buf, "ori.b␣#$%X,d%X", imm.b, reg );
            address += 4;
        }
        if( size == 1 )
        {
            if( decImm )sprintf( buf, "ori.w␣#%d,d%X", imm.w, reg );
            else sprintf( buf, "ori.w␣#$%X,d%X", imm.w, reg );
            address += 4;
        }
        if( size == 2 )
        {
            if( decImm )sprintf( buf, "ori.l␣#%ld,d%X", imm.l, reg );
```

```
        else sprintf( buf, " ori.l #$%lX,d%X", imm.l, reg );
        address += 6;
    }
}

// #data,(an)
if( mode == 2 )
{
    if( size == 0 )
    {
        if( decImm )sprintf( buf, " ori.b #%d,(a%X)", imm.b, reg );
        else sprintf( buf, " ori.b #$%X,(a%X)", imm.b, reg );
        address += 4;
    }
    if( size == 1 )
    {
        if( decImm )sprintf( buf, " ori.w #%d,(a%X)", imm.w, reg );
        else sprintf( buf, " ori.w #$%X,(a%X)", imm.w, reg );
        address += 4;
    }
    if( size == 2 )
    {
        if( decImm )sprintf( buf, " ori.l #%ld,(a%X)", imm.l, reg );
        else sprintf( buf, " ori.l #$%lX,(a%X)", imm.l, reg );
        address += 6;
    }
}

// #data,(an)+
if( mode == 3 )
{
    if( size == 0 )
    {
        if( decImm )sprintf( buf, " ori.b #%d,(a%X)+", imm.b, reg );
        else sprintf( buf, " ori.b #$%X,(a%X)+", imm.b, reg );
        address += 4;
    }
    if( size == 1 )
    {
        if( decImm )sprintf( buf, " ori.w #%d,(a%X)+", imm.w, reg );
        else sprintf( buf, " ori.w #$%X,(a%X)+", imm.w, reg );
        address += 4;
    }
    if( size == 2 )
    {
        if( decImm )sprintf( buf, " ori.l #%ld,(a%X)+", imm.l, reg );
        else sprintf( buf, " ori.l #$%lX,(a%X)+", imm.l, reg );
        address += 6;
    }
}

// #data,-(an)
```

```
if( mode == 4 )
{
    if( size == 0 )
    {
        if( decImm )sprintf( buf, " ori.b␣#%d,-(a%X)", imm.b, reg );
        else sprintf( buf, " ori.b␣#$%X,-(a%X)", imm.b, reg );
        address += 4;
    }
    if( size == 1 )
    {
        if( decImm )sprintf( buf, " ori.w␣#%d,-(a%X)", imm.w, reg );
        else sprintf( buf, " ori.w␣#$%X,-(a%X)", imm.w, reg );
        address += 4;
    }
    if( size == 2 )
    {
        if( decImm )sprintf( buf, " ori.l␣#%ld,-(a%X)", imm.l, reg );
        else sprintf( buf, " ori.l␣#$%lX,-(a%X)", imm.l, reg );
        address += 6;
    }
}

// #data,d16(an)
if( mode == 5 )
{
    if( size == 0 )
    {
        displ = (signed short int)memory->GetWord( address + 4 );
        if( decImm )sprintf( buf, " ori.b␣#$%X,%ld(a%X)", imm.b, displ, reg );
        else sprintf( buf, " ori.b␣#$%X,$%lX(a%X)", imm.b, displ, reg );
        address += 6;
    }
    if( size == 1 )
    {
        displ = (signed short int)memory->GetWord( address + 4 );
        if( decImm )sprintf( buf, " ori.w␣#$%X,%ld(a%X)", imm.w, displ, reg );
        else sprintf( buf, " ori.w␣#$%X,$%lX(a%X)", imm.w, displ, reg );
        address += 6;
    }
    if( size == 2 )
    {
        displ = (signed short int)memory->GetWord( address + 6 )+ a[ reg ].l;
        if( decImm )sprintf( buf, " ori.l␣#$%lX,%ld(a%X)", imm.l, displ, reg );
        else sprintf( buf, " ori.l␣#$%lX,$%lX(a%X)", imm.l, displ, reg );
        address += 8;
    }
}

// #data,d8(an,xn)
if( mode == 6 )
{
    if( size == 0 )
```

```
    {
        displ = (signed char)memory->GetByte( address + 5 );
        xn = GETBITS( address + 4, 0xF000, 12 );
        if ( decImm )sprintf ( buf, " ori.b #$%X,%ld(a%X,d%X)", imm.b, displ, reg, xn );
        else sprintf ( buf, " ori.b #$%X,$%lX(a%X,d%X)", imm.b, displ, reg, xn );
        address += 6;
    }
    if ( size == 1 )
    {
        displ = (signed char)memory->GetByte( address + 5 );
        xn = GETBITS( address + 4, 0xF000, 12 );
        if ( decImm )sprintf ( buf, " ori.w #$%X,%ld(a%X,d%X)", imm.w, displ, reg, xn );
        else sprintf ( buf, " ori.w #$%X,$%lX(a%X,d%X)", imm.w, displ, reg, xn );
        address += 6;
    }
    if ( size == 2 )
    {
        displ = (signed char)memory->GetByte( address + 7 );
        xn = GETBITS( address + 6, 0xF000, 12 );
        if ( decImm )sprintf ( buf, " ori.l #$%lX,%ld(a%X,d%X)", imm.l, displ, reg, xn );
        else sprintf ( buf, " ori.l #$%lX,$%lX(a%X,d%X)", imm.l, displ, reg, xn );
        address += 8;
    }
}

if ( mode == 7 )
{
    // #data,(xxx).w
    if ( reg == 0 )
    {
        if ( size == 0 )
        {
            displ = memory->GetWord( address + 4 );
            if ( decImm )sprintf ( buf, " ori.b #%d,$%lX", imm.b, displ );
            else sprintf ( buf, " ori.b #$%X,$%lX", imm.b, displ );
            address += 6;
        }
        if ( size == 1 )
        {
            displ = memory->GetWord( address + 4 );
            if ( decImm )sprintf ( buf, " ori.w #%d,$%lX", imm.w, displ );
            else sprintf ( buf, " ori.w #$%X,$%lX", imm.w, displ );
            address += 6;
        }
        if ( size == 2 )
        {
            displ = memory->GetWord( address + 6 );
            if ( decImm )sprintf ( buf, " ori.l #%ld,$%lX", imm.l, displ );
            else sprintf ( buf, " ori.l #$%lX,$%lX", imm.l, displ );
            address += 8;
        }
    }
```

```
        // #data,(xxx).l
        if( reg == 1 )
        {
            if( size == 0 )
            {
                displ = memory->GetLongword( address + 4 );
                if( decImm )sprintf( buf, " ori.b #%d,$%lX", imm.b, displ );
                else sprintf( buf, " ori.b #$%X,$%lX", imm.b, displ );
                address += 8;
            }
            if( size == 1 )
            {
                displ = memory->GetLongword( address + 4 );
                if( decImm )sprintf( buf, " ori.w #%d,$%lX", imm.w, displ );
                else sprintf( buf, " ori.w #$%X,$%lX", imm.w, displ );
                address += 8;
            }
            if( size == 2 )
            {
                displ = memory->GetLongword( address + 6 );
                if( decImm )sprintf( buf, " ori.l #%ld,$%lX", imm.l, displ );
                else sprintf( buf, " ori.l #$%lX,$%lX", imm.l, displ );
                address += 10;
            }
        }
    }

    return address;
}

unsigned long M68008::tS_oriCcr( unsigned long address, char *buf )
{
    ccr.x |= GETBITS( address + 2, 0x0010, 4 );
    ccr.n |= GETBITS( address + 2, 0x0008, 3 );
    ccr.z |= GETBITS( address + 2, 0x0004, 2 );
    ccr.v |= GETBITS( address + 2, 0x0002, 1 );
    ccr.c |= GETBITS( address + 2, 0x0001, 0 );

    if( decImm )sprintf( buf, " ori.b #%d,ccr", (unsigned char)memory->GetByte( address + 3 ));
    else sprintf( buf, " ori.b #$%X,ccr", (unsigned char)memory->GetByte( address + 3 ));

    address += 4;

    return address;
}

unsigned long M68008::tS_pea( unsigned long address, char *buf )
{
    int mode, reg;

    mode = GETBITS( address, 0x0038, 3 );
```

```
    reg   = GETBITS( address, 0x0007, 0 );

    a [ 7 ]. l -= 4;

    // (an)
    if ( mode == 2 )
    {
        sprintf ( buf, "pea.l (a%X)", reg );
        address += 2;
    }

    // d16(an)
    if ( mode == 5 )
    {
        if ( decImm )sprintf ( buf, "pea.l %d(a%X)", (signed short int)memory->GetWord( address + 2 ), reg
            );
        else sprintf ( buf, "pea.l $%X(a%X)", (signed short int)memory->GetWord( address + 2 ), reg );
        address += 4;
    }

    if ( mode == 6 )
    // d8(an,xn)
    {
        if ( decImm )sprintf ( buf, "pea.l %d(a%X,d%X", (signed char)memory->GetByte( address + 3 ), reg,
            GETBITS( address + 2, 0xF000, 12 ));
        else sprintf ( buf, "pea.l $%X(a%X,d%X", (signed char)memory->GetByte( address + 3 ), reg,
            GETBITS( address + 2, 0xF000, 12 ));
        address += 4;
    }

    // (xxx).w
    if ( ( mode == 7 )&& ( reg == 0 ))
    {
        sprintf ( buf, "pea.l $%X", memory->GetWord( address + 2 ));
        address += 4;
    }

    // (xxx).l
    if ( ( mode == 7 )&& ( reg == 1 ))
    {
        sprintf ( buf, "pea.l $%lX", memory->GetLongword( address + 2 ));
        address += 6;
    }

    return address;
}

unsigned long M68008::tS_rol_rReg( unsigned long address, char *buf )
{
    int cnt, dr, size, ir, reg;

    cnt   = GETBITS( address, 0x0E00, 9 );
```

```
dr   = GETBITS( address, 0x0100, 8 );
size = GETBITS( address, 0x00C0, 6 );
ir   = GETBITS( address, 0x0020, 5 );
reg  = GETBITS( address, 0x0007, 0 );

// rotate right
if( dr == 0 )
{
    // #data,dn
    if( ir == 0 )
    {
        if( !cnt ) cnt = 8;

        // byte
        if( size == 0 )
        {
            if( decImm )sprintf( buf, "ror.b #%d,d%X", cnt, reg );
            else sprintf( buf, "ror.b #$%X,d%X", cnt, reg );
        }

        // word
        if( size == 1 )
        {
            if( decImm )sprintf( buf, "ror.w #%d,d%X", cnt, reg );
            else sprintf( buf, "ror.w #$%X,d%X", cnt, reg );
        }

        // longword
        if( size == 2 )
        {
            if( decImm )sprintf( buf, "ror.l #%d,d%X", cnt, reg );
            else sprintf( buf, "ror.l #$%X,d%X", cnt, reg );
        }
    }

    // dn,dn
    if( ir == 1 )
    {
        // byte
        if( size == 0 )
        {
            sprintf( buf, "ror.b d%X,d%X", cnt, reg );
        }

        // word
        if( size == 1 )
        {
            sprintf( buf, "ror.w d%X,d%X", cnt, reg );
        }

        // longword
        if( size == 2 )
```

```
        {
            sprintf ( buf, " ror.l d%X,d%X", cnt, reg );
        }
    }
}

// rotate  left
if ( dr == 1 )
{
    // #data,dn
    if ( ir == 0 )
    {
        if ( !cnt ) cnt = 8;

        // byte
        if ( size == 0 )
        {
            if ( decImm )sprintf ( buf, " rol.b #%d,d%X", cnt, reg );
            else sprintf ( buf, " rol.b #$%X,d%X", cnt, reg );
        }

        // word
        if ( size == 1 )
        {
            if ( decImm )sprintf ( buf, " rol.w #%d,d%X", cnt, reg );
            else sprintf ( buf, " rol.w #$%X,d%X", cnt, reg );
        }

        // longword
        if ( size == 2 )
        {
            if ( decImm )sprintf ( buf, " rol.l #%d,d%X", cnt, reg );
            else sprintf ( buf, " rol.l #$%X,d%X", cnt, reg );
        }
    }

    // dn,dn
    if ( ir == 1 )
    {
        // byte
        if ( size == 0 )
        {
            sprintf ( buf, " rol.b d%X,d%X", cnt, reg );
        }

        // word
        if ( size == 1 )
        {
            sprintf ( buf, " rol.w d%X,d%X", cnt, reg );
        }

        // longword
```

```
            if ( size == 2 )
            {
                sprintf ( buf, " rol.l d%X,d%X", cnt, reg );
            }
        }
    }

    address += 2;

    return address;
}

unsigned long M68008::tS_rol_rMem( unsigned long address, char *buf )
{
    int mode, reg, dr, xn;
    unsigned long displ;

    dr   = GETBITS( address, 0x0100, 8 );
    mode = GETBITS( address, 0x0038, 3 );
    reg  = GETBITS( address, 0x0007, 0 );

    // (an)
    if ( mode == 2 )
    {
        // right
        if ( dr == 0 )
        {
            sprintf ( buf, " ror.w (a%X)", reg );
        }

        // left
        if ( dr == 1 )
        {
            sprintf ( buf, " rol.w (a%X)", reg );
        }

        address += 2;
    }

    // (an)+
    if ( mode == 3 )
    {
        // right
        if ( dr == 0 )
        {
            sprintf ( buf, " ror.w (a%X)+", reg );
        }

        // left
        if ( dr == 1 )
        {
            sprintf ( buf, " rol.w (a%X)+", reg );
```

```
    }

    address += 2;
}

// -(an)
if( mode == 4 )
{
    // right
    if( dr == 0 )
    {
        sprintf( buf, "ror.w -(a%X)", reg );
    }

    // left
    if( dr == 1 )
    {
        sprintf( buf, "rol.w -(a%X)", reg );
    }

    address += 2;
}

// d16(an)
if( mode == 5 )
{
    displ = memory->GetWord( address + 2 );

    // right
    if( dr == 0 )
    {
        if( decImm )sprintf( buf, "ror.w %ld(a%X)", displ, reg );
        else sprintf( buf, "ror.w $%lX(a%X)", displ, reg );
    }

    // left
    if( dr == 1 )
    {
        if( decImm )sprintf( buf, "rol.w %ld(a%X)", displ, reg );
        else sprintf( buf, "rol.w $%lX(a%X)", displ, reg );
    }

    address += 4;
}

// d8(an,xn)
if( mode == 6 )
{
    displ = memory->GetByte( address + 3 );
    xn = GETBITS( address + 2, 0xF000, 12 );

    // right
```

```
        if( dr == 0 )
        {
            if( decImm )sprintf( buf, "ror.w␣%ld(a%X,d%X)", displ, reg, xn );
            else sprintf( buf, "ror.w␣$%lX(a%X,d%X)", displ, reg, xn );
        }

        // left
        if( dr == 1 )
        {
            if( decImm )sprintf( buf, "rol.w␣%ld(a%X,d%X)", displ, reg, xn );
            else sprintf( buf, "rol.w␣$%lX(a%X,d%X)", displ, reg, xn );
        }

        address += 4;
    }

    if( mode == 7 )
    {
        // (xxx).w
        if( reg == 0 )
        {
            displ = memory->GetWord( address + 2 );
            address += 4;
        }
        else
        {
            displ = memory->GetLongword( address + 2 );
            address += 6;
        }

        // right
        if( dr == 0 )
        {
            sprintf( buf, "ror.w␣$%lX", displ );
        }

        // left
        if( dr == 1 )
        {
            sprintf( buf, "rol.w␣$%lX", displ );
        }
    }

    return address;
}

unsigned long M68008::tS_roxl_rReg( unsigned long address, char *buf )
{
    int cnt, dr, size, ir, reg;

    cnt  = GETBITS( address, 0x0E00, 9 );
    dr   = GETBITS( address, 0x0100, 8 );
```

```
size = GETBITS( address, 0x00C0, 6 );
ir   = GETBITS( address, 0x0020, 5 );
reg  = GETBITS( address, 0x0007, 0 );

// right
if( dr == 0 )
{
   // #data
   if( ir == 0 )
   {
      // byte
      if( size == 0 )
      {
         if( decImm )sprintf( buf, "roxr.b #%d,d%X", cnt, reg );
         else sprintf( buf, "roxr.b #$%X,d%X", cnt, reg );
      }

      // word
      if( size == 1 )
      {
         if( decImm )sprintf( buf, "roxr.w #%d,d%X", cnt, reg );
         else sprintf( buf, "roxr.w #$%X,d%X", cnt, reg );
      }

      // longword
      if( size == 2 )
      {
         if( decImm )sprintf( buf, "roxr.l #%d,d%X", cnt, reg );
         else sprintf( buf, "roxr.l #$%X,d%X", cnt, reg );
      }
   }

   // dn
   if( ir == 1 )
   {
      // byte
      if( size == 0 )
      {
         sprintf( buf, "roxr.b d%X,d%X", cnt, reg );
      }

      // word
      if( size == 1 )
      {
         sprintf( buf, "roxr.w d%X,d%X", cnt, reg );
      }

      // longword
      if( size == 2 )
      {
         sprintf( buf, "roxr.l d%X,d%X", cnt, reg );
      }
```

```
    }
}

// left
if( dr == 1 )
{
    // #data
    if( ir == 0 )
    {
        // byte
        if( size == 0 )
        {
            if( decImm )sprintf( buf, " roxl.b #%d,d%X", cnt, reg );
            else sprintf( buf, " roxl.b #$%X,d%X", cnt, reg );
        }

        // word
        if( size == 1 )
        {
            if( decImm )sprintf( buf, " roxl.w #%d,d%X", cnt, reg );
            else sprintf( buf, " roxl.w #$%X,d%X", cnt, reg );
        }

        // longword
        if( size == 2 )
        {
            if( decImm )sprintf( buf, " roxl.l #%d,d%X", cnt, reg );
            else sprintf( buf, " roxl.l #$%X,d%X", cnt, reg );
        }
    }

    // dn
    if( ir == 1 )
    {
        // byte
        if( size == 0 )
        {
            sprintf( buf, " roxl.b d%X,d%X", cnt, reg );
        }

        // word
        if( size == 1 )
        {
            sprintf( buf, " roxl.w d%X,d%X", cnt, reg );
        }

        // longword
        if( size == 2 )
        {
            sprintf( buf, " roxl.l d%X,d%X", cnt, reg );
        }
    }
```

```
    }

    address += 2;

    return address;
}

unsigned long M68008::tS_roxl_rMem( unsigned long address, char *buf )
{
    int mode, reg, dr, xn;
    unsigned long displ;

    dr   = GETBITS( address, 0x0100, 8 );
    mode = GETBITS( address, 0x0038, 3 );
    reg  = GETBITS( address, 0x0007, 0 );

    // (an)
    if( mode == 2 )
    {
        // right
        if( dr == 0 )
        {
            sprintf( buf, "roxr.w (a%X)", reg );
        }

        // left
        if( dr == 1 )
        {
            sprintf( buf, "roxl.w (a%X)", reg );
        }

        address += 2;
    }

    // (an)+
    if( mode == 3 )
    {
        // right
        if( dr == 0 )
        {
            sprintf( buf, "roxr.w (a%X)+", reg );
        }

        // left
        if( dr == 1 )
        {
            sprintf( buf, "roxl.w (a%X)+", reg );
        }

        address += 2;
    }
```

```
// -(an)
if( mode == 4 )
{
    // right
    if( dr == 0 )
    {
        sprintf( buf, "roxr.w -(a%X)", reg );
    }

    // left
    if( dr == 1 )
    {
        sprintf( buf, "roxl.w -(a%X)", reg );
    }

    address += 2;
}

// d16(an)
if( mode == 5 )
{
    displ = memory->GetWord( address + 2 );

    // right
    if( dr == 0 )
    {
        if( decImm )sprintf( buf, "roxr.w %ld(a%X)", displ, reg );
        else sprintf( buf, "roxr.w $%lX(a%X)", displ, reg );
    }

    // left
    if( dr == 1 )
    {
        if( decImm )sprintf( buf, "roxl.w %ld(a%X)", displ, reg );
        else sprintf( buf, "roxl.w $%lX(a%X)", displ, reg );
    }

    address += 4;
}

// d8(an,xn)
if( mode == 6 )
{
    displ = memory->GetByte( address + 3 );
    xn = GETBITS( address + 2, 0xF000, 12 );

    // right
    if( dr == 0 )
    {
        if( decImm )sprintf( buf, "roxr.w %ld(a%X,d%X)", displ, reg, xn );
        else sprintf( buf, "roxr.w $%lX(a%X,d%X)", displ, reg, xn );
    }
```

```
        // left
        if( dr == 1 )
        {
            if( decImm )sprintf( buf, "roxl.w #%ld(a%X,d%X)", displ, reg, xn );
            else sprintf( buf, "roxl.w #$%lX(a%X,d%X)", displ, reg, xn );
        }

        address += 4;
    }

    if( mode == 7 )
    {
        // (xxx).w
        if( reg == 0 )
        {
            displ = memory->GetWord( address + 2 );
            address += 4;
        }
        else
        {
            displ = memory->GetLongword( address + 2 );
            address += 6;
        }

        // right
        if( dr == 0 )
        {
            sprintf( buf, "roxr.w #$%lX", displ );
        }

        // left
        if( dr == 1 )
        {
            sprintf( buf, "roxl.w #$%lX", displ );
        }
    }

    return address;
}

unsigned long M68008::tS_rts( unsigned long address, char *buf )
{
    sprintf( buf, "rts" );

    return address + 2;
}

unsigned long M68008::tS_sbcd( unsigned long address, char *buf )
{
    sprintf( buf, "sbcd: Unimplemented" );
```

```
    return address + 2;
}

unsigned long M68008::tS_sxx( unsigned long address, char *buf )
{
    int cond, mode, reg, xn;
    unsigned long displ = 0;

    cond = GETBITS( address, 0x0F00, 8 );
    mode = GETBITS( address, 0x0038, 3 );
    reg  = GETBITS( address, 0x0007, 0 );

    char cmd[ 4 ];

    if( cond == 0 ) sprintf( cmd, "st" );
    if( cond == 1 ) sprintf( cmd, "sf" );
    if( cond == 2 ) sprintf( cmd, "shi" );
    if( cond == 3 ) sprintf( cmd, "sls" );
    if( cond == 4 ) sprintf( cmd, "scc" );
    if( cond == 5 ) sprintf( cmd, "scs" );
    if( cond == 6 ) sprintf( cmd, "sne" );
    if( cond == 7 ) sprintf( cmd, "seq" );
    if( cond == 8 ) sprintf( cmd, "svc" );
    if( cond == 9 ) sprintf( cmd, "svs" );
    if( cond == 10 )sprintf( cmd, "spl" );
    if( cond == 11 )sprintf( cmd, "smi" );
    if( cond == 12 )sprintf( cmd, "sge" );
    if( cond == 13 )sprintf( cmd, "slt" );
    if( cond == 14 )sprintf( cmd, "sgt" );
    if( cond == 15 )sprintf( cmd, "sle" );

    // dn
    if( mode == 0 )
    {
        sprintf( buf, "%s.b d%X", cmd, reg );
        address += 2;
    }

    // (an)
    if( mode == 2 )
    {
        sprintf( buf, "%s.b (a%X)", cmd, reg );
        address += 2;
    }

    // (an)+
    if( mode == 3 )
    {
        sprintf( buf, "%s.b (a%X)+", cmd, reg );
        address += 2;
    }
```

```
    // -(an)
    if ( mode == 4 )
    {
        sprintf ( buf, "%s.b -(a%X)", cmd, reg );
        address += 2;
    }

    // d16(an)
    if ( mode == 5 )
    {
        displ = memory->GetWord( address + 2 );
        if ( decImm )sprintf ( buf, "%s.b %ld(a%X)", cmd, displ, reg );
        else sprintf ( buf, "%s.b $%lX(a%X)", cmd, displ, reg );
        address += 4;
    }

    // d8(an,xn)
    if ( mode == 6 )
    {
        displ = memory->GetByte( address + 3 );
        xn = GETBITS( address + 2, 0xF000, 12 );
        if ( decImm )sprintf ( buf, "%s.b %ld(a%X,d%X)", cmd, displ, reg, xn );
        else sprintf ( buf, "%s.b $%lX(a%X,d%X)", cmd, displ, reg, xn );
        address += 4;
    }

    if ( mode == 7 )
    {
        // (xxx).w
        if ( reg == 0 )
        {
            displ = memory->GetWord( address + 2 );
            address += 4;
        }

        // (xxx).l
        if ( reg == 1 )
        {
            displ = memory->GetLongword( address + 2 );
            address += 6;
        }

        sprintf ( buf, "%s.b $%lX", cmd, displ );
    }

    return address;
}

unsigned long M68008::tS_sub( unsigned long address, char *buf )
{
    int mode, opmode, reg, eaReg, xn;
    unsigned long addr;
```

```
reg    = GETBITS( address, 0x0E00, 9 );
opmode = GETBITS( address, 0x01C0, 6 );
mode   = GETBITS( address, 0x0038, 3 );
eaReg  = GETBITS( address, 0x0007, 0 );

if ( ( opmode & 3 )== 3 )
{
   return tS_suba( address, buf );
}

// dn,dn
if ( mode == 0 )
{
   if ( opmode == 0 )
   {
      sprintf ( buf, "sub.b d%X,d%X", eaReg, reg );
   }
   if ( opmode == 1 )
   {
      sprintf ( buf, "sub.w d%X,d%X", eaReg, reg );
   }
   if ( opmode == 2 )
   {
      sprintf ( buf, "sub.l d%X,d%X", eaReg, reg );
   }
   if ( opmode == 4 )
   {
      sprintf ( buf, "sub.b d%X,d%X", reg, eaReg );
   }
   if ( opmode == 5 )
   {
      sprintf ( buf, "sub.b d%X,d%X", reg, eaReg );
   }
   if ( opmode == 6 )
   {
      sprintf ( buf, "sub.b d%X,d%X", reg, eaReg );
   }
   address += 2;
}

// an,dn ( dn - an )
if ( mode == 1 )
{
   if ( opmode == 1 )
   {
      sprintf ( buf, "sub.w a%X,d%X", eaReg, reg );
   }
   if ( opmode == 2 )
   {
      sprintf ( buf, "sub.l a%X,d%X", eaReg, reg );
   }
```

```
    address += 2;
}

// (an),dn ( dn - (an) )
if( mode == 2 )
{
    if( opmode == 0 )
    {
        sprintf( buf, "sub.b (a%X),d%X", eaReg, reg );
    }
    if( opmode == 1 )
    {
        sprintf( buf, "sub.w (a%X),d%X", eaReg, reg );
    }
    if( opmode == 2 )
    {
        sprintf( buf, "sub.l (a%X),d%X", eaReg, reg );
    }
    if( opmode == 4 )
    {
        sprintf( buf, "sub.b d%X,(a%X)", reg, eaReg );
    }
    if( opmode == 5 )
    {
        sprintf( buf, "sub.w d%X,(a%X)", reg, eaReg );
    }
    if( opmode == 6 )
    {
        sprintf( buf, "sub.l d%X,(a%X)", reg, eaReg );
    }

    address += 2;
}

// (an)+,dn ( dn - (an) )
if( mode == 3 )
{
    if( opmode == 0 )
    {
        sprintf( buf, "sub.b (a%X)+,d%X", eaReg, reg );
    }
    if( opmode == 1 )
    {
        sprintf( buf, "sub.w (a%X)+,d%X", eaReg, reg );
    }
    if( opmode == 2 )
    {
        sprintf( buf, "sub.l (a%X)+,d%X", eaReg, reg );
    }
    if( opmode == 4 )
    {
        sprintf( buf, "sub.b d%X,(a%X)+", reg, eaReg );
```

```
        }
        if ( opmode == 5 )
        {
            sprintf ( buf, "sub.w d%X,(a%X)+", reg, eaReg );
        }
        if ( opmode == 6 )
        {
            sprintf ( buf, "sub.l d%X,(a%X)+", reg, eaReg );
        }

        address += 2;
    }

    // -(an),dn ( dn - (an) )
    if ( mode == 4 )
    {
        if ( opmode == 0 )
        {
            sprintf ( buf, "sub.b -(a%X),d%X", eaReg, reg );
        }
        if ( opmode == 1 )
        {
            sprintf ( buf, "sub.w -(a%X),d%X", eaReg, reg );
        }
        if ( opmode == 2 )
        {
            sprintf ( buf, "sub.l -(a%X),d%X", eaReg, reg );
        }
        if ( opmode == 4 )
        {
            sprintf ( buf, "sub.b d%X,-(a%X)", reg, eaReg );
        }
        if ( opmode == 5 )
        {
            sprintf ( buf, "sub.w d%X,-(a%X)", reg, eaReg );
        }
        if ( opmode == 6 )
        {
            sprintf ( buf, "sub.l d%X,-(a%X)", reg, eaReg );
        }

        address += 2;

    }

    // d16(an),dn ( dn - d16(an) )
    if ( mode == 5 )
    {
        unsigned long addr = (signed short int)memory->GetWord( address + 2 );
        if ( opmode == 0 )
        {
            if ( decImm )sprintf ( buf, "sub.b %ld(a%X),d%X", addr, eaReg, reg );
```

```
        else sprintf( buf, "sub.b␣$%lX(a%X),d%X", addr, eaReg, reg );
    }
    if( opmode == 1 )
    {
        if( decImm )sprintf( buf, "sub.w␣%ld(a%X),d%X", addr, eaReg, reg );
        else sprintf( buf, "sub.w␣$%lX(a%X),d%X", addr, eaReg, reg );
    }
    if( opmode == 2 )
    {
        if( decImm )sprintf( buf, "sub.l␣%ld(a%X),d%X", addr, eaReg, reg );
        else sprintf( buf, "sub.l␣$%lX(a%X),d%X", addr, eaReg, reg );
    }
    if( opmode == 4 )
    {
        if( decImm )sprintf( buf, "sub.l␣d%X,%ld(a%X)", reg, addr, eaReg );
        else sprintf( buf, "sub.l␣d%X,$%lX(a%X)", reg, addr, eaReg );
    }
    if( opmode == 5 )
    {
        if( decImm )sprintf( buf, "sub.l␣d%X,%ld(a%X)", reg, addr, eaReg );
        else sprintf( buf, "sub.l␣d%X,$%lX(a%X)", reg, addr, eaReg );
    }
    if( opmode == 6 )
    {
        if( decImm )sprintf( buf, "sub.l␣d%X,%ld(a%X)", reg, addr, eaReg );
        else sprintf( buf, "sub.l␣d%X,$%lX(a%X)", reg, addr, eaReg );
    }
    address += 4;
}

// d8(an,Xn),dn ( dn - d8(an) )
if( mode == 6 )
{
    addr = (signed char)memory->GetByte( address + 3 );
    xn = GETBITS( address + 2, 0xF000, 12 );

    if( opmode == 0 )
    {
        if( decImm )sprintf( buf, "sub.b␣%ld(a%X,d%X),d%X", addr, eaReg, xn, reg );
        else sprintf( buf, "sub.b␣$%lX(a%X,d%X),d%X", addr, eaReg, xn, reg );
    }
    if( opmode == 1 )
    {
        if( decImm )sprintf( buf, "sub.w␣%ld(a%X,d%X),d%X", addr, eaReg, xn, reg );
        else sprintf( buf, "sub.w␣$%lX(a%X,d%X),d%X", addr, eaReg, xn, reg );
    }
    if( opmode == 2 )
    {
        if( decImm )sprintf( buf, "sub.l␣%ld(a%X,d%X),d%X", addr, eaReg, xn, reg );
        else sprintf( buf, "sub.l␣$%lX(a%X,d%X),d%X", addr, eaReg, xn, reg );
    }
    if( opmode == 4 )
```

```
    {
        if ( decImm )sprintf ( buf, "sub.b␣d%X,%ld(a%X,d%X)", reg, addr, eaReg, xn );
        else  sprintf ( buf, "sub.b␣d%X,$%lX(a%X,d%X)", reg, addr, eaReg, xn );
    }
    if ( opmode == 5 )
    {
        if ( decImm )sprintf ( buf, "sub.w␣d%X,%ld(a%X,d%X)", reg, addr, eaReg, xn );
        else  sprintf ( buf, "sub.w␣d%X,$%lX(a%X,d%X)", reg, addr, eaReg, xn );
    }
    if ( opmode == 6 )
    {
        if ( decImm )sprintf ( buf, "sub.l␣d%X,%ld(a%X,d%X)", reg, addr, eaReg, xn );
        else  sprintf ( buf, "sub.l␣d%X,$%lX(a%X,d%X)", reg, addr, eaReg, xn );
    }
    address += 4;
}

//(XXX).W,dn ( dn − (XXX).W − dn) )
if ( mode == 7 )
{
    if ( eaReg == 0 )
    {
        addr = memory->GetWord( address + 2 );
        address += 4;
    }
    else
    {
        addr = memory->GetLongword( address + 2 );
        address += 6;
    }
    if ( opmode == 0 )
    {
        sprintf ( buf, "sub.b␣$%lX,d%X", addr, reg );
    }
    if ( opmode == 1 )
    {
        sprintf ( buf, "sub.w␣$%lX,d%X", addr, reg );
    }
    if ( opmode == 2 )
    {
        sprintf ( buf, "sub.l␣$%lX,d%X", addr, reg );
    }
    if ( opmode == 4 )
    {
        sprintf ( buf, "sub.b␣d%X,$%lX", reg, addr );
    }
    if ( opmode == 5 )
    {
        sprintf ( buf, "sub.w␣d%X,$%lX", reg, addr );
    }
    if ( opmode == 6 )
    {
```

```
            sprintf ( buf, "sub.l d%X,$%lX", reg, addr );
        }
    }

    // OMITTED
    // #< data >, dn ( see subi )
    // ( D16, PC )
    // ( d8, PC, Xn )

    return address;
}

unsigned long M68008::tS_suba( unsigned long address, char *buf )
{
    int mode, opmode, reg, eaReg, xn;
    unsigned long addr;
    ADDRESS_REGISTER( imm );

    reg = GETBITS( address, 0x0E00, 9 );
    opmode = GETBITS( address, 0x01C0, 6 );
    mode = GETBITS( address, 0x0038, 3 );
    eaReg = GETBITS( address, 0x0007, 0 );

    // dn,an ( an - dn )
    if ( mode == 0 )
    {
        if ( opmode == 3 )
        {
            sprintf ( buf, "suba.w d%X,a%X", eaReg, reg );
        }
        if ( opmode == 7 )
        {
            sprintf ( buf, "suba.l d%X,a%X", eaReg, reg );
        }
        address += 2;
    }
    // an, an
    if ( mode == 1 )
    {
        if ( opmode == 3 )
        {
            sprintf ( buf, "suba.w a%X,a%X", eaReg, reg );
        }
        if ( opmode == 7 )
        {
            sprintf ( buf, "suba.l a%X,a%X", eaReg, reg );
        }
        address += 2;
    }
    // ( an ), an
    if ( mode == 2 )
    {
```

```
    if ( opmode == 3 )
    {
        sprintf ( buf, "suba.w (a%X),a%X", eaReg, reg );
    }
    if ( opmode == 7 )
    {
        sprintf ( buf, "suba.l (a%X),a%X", eaReg, reg );
    }
    address += 2;
}
// ( an )+, an
if ( mode == 3 )
{
    if ( opmode == 3 )
    {
        sprintf ( buf, "suba.w (a%X)+,a%X", eaReg, reg );
    }
    if ( opmode == 7 )
    {
        sprintf ( buf, "suba.l (a%X)+,a%X", eaReg, reg );
    }
    address += 2;
}
// -( an ), an
if ( mode == 4 )
{
    if ( opmode == 3 )
    {
        sprintf ( buf, "suba.w -(a%X),a%X", eaReg, reg );
    }
    if ( opmode == 7 )
    {
        sprintf ( buf, "suba.l -(a%X),a%X", eaReg, reg );
    }
    address += 2;
}
// d16(an), an
if ( mode == 5 )
{
    addr = (signed short int)memory->GetWord( address + 2 );
    if ( opmode == 3 )
    {
        if ( decImm )sprintf ( buf, "suba.w %ld(a%X),a%X", addr, eaReg, reg );
        else  sprintf ( buf, "suba.w $%lX(a%X),a%X", addr, eaReg, reg );
    }
    if ( opmode == 7 )
    {
        if ( decImm )sprintf ( buf, "suba.l %ld(a%X),a%X", addr, eaReg, reg );
        else  sprintf ( buf, "suba.l $%lX(a%X),a%X", addr, eaReg, reg );
    }
    address += 4;
}
```

```
// d8(an,xn), an
if ( mode == 6 )
{
    addr = (signed char)memory->GetByte( address + 3 );
    xn = GETBITS( address + 2, 0xf000, 12 );
    if ( opmode == 3 )
    {
        if ( decImm )sprintf ( buf, "suba.w %ld(a%X,d%X),a%X", addr, eaReg, xn, reg );
        else  sprintf ( buf, "suba.w $%lX(a%X,d%X),a%X", addr, eaReg, xn, reg );
    }
    if ( opmode == 7 )
    {
        if ( decImm )sprintf ( buf, "suba.l %ld(a%X,d%X),a%X", addr, eaReg, xn, reg );
        else  sprintf ( buf, "suba.l $%lX(a%X,d%X),a%X", addr, eaReg, xn, reg );
    }
    address += 4;
}

// #<data>,an
if ( ( mode == 7 )&& ( eaReg == 4 ))
{
    if ( opmode == 3 )
    {
        imm.w = (signed short int)memory->GetWord( address + 2 );
        if ( decImm )sprintf ( buf, "suba.w #%d,a%X", imm.w, reg );
        else  sprintf ( buf, "suba.w #$%X,a%X", imm.w, reg );
        address += 4;
    }
    if ( opmode == 7 )
    {
        imm.l = (signed long int)memory->GetLongword( address + 2 );
        if ( decImm )sprintf ( buf, "suba.l #%ld,a%X", imm.l, reg );
        else  sprintf ( buf, "suba.l #$%lX,a%X", imm.l, reg );
        address += 6;
    }
    return address;
}

//(XXX).W, an
if ( mode == 7 )
{
    if ( eaReg == 0 )
    {
        addr = memory->GetWord( address + 2 );
        address += 4;
    }
    else
    {
        //(XXX).L, an
        addr = memory->GetLongword( address + 2 );
        address += 6;
    }
```

```
        if ( opmode == 3 )
        {
            sprintf ( buf, "suba.w $%lX,a%X", addr, reg );
        }
        if ( opmode == 7 )
        {
            sprintf ( buf, "suba.l $%lX,a%X", addr, reg );
        }
    }

    return address;
}

unsigned long M68008::tS_subi( unsigned long address, char *buf )
{
    int  size , mode, reg, displ , xn;
    DATA_REGISTER( imm );

    size = GETBITS( address, 0x00C0, 6 );
    mode = GETBITS( address, 0x0038, 3 );
    reg = GETBITS( address, 0x0007, 0 );

    // #data,dn
    if ( mode == 0 )
    {
        if ( size == 0 )
        {
            imm.b = memory->GetByte( address + 3 );
            if ( decImm )sprintf ( buf, "subi.b #%d,d%X", imm.b, reg );
            else  sprintf ( buf, "subi.b #$%X,d%X", imm.b, reg );
            address += 4;
        }
        if ( size == 1 )
        {
            imm.w = memory->GetWord( address + 2 );
            if ( decImm )sprintf ( buf, "subi.w #%d,d%X", imm.w, reg );
            else  sprintf ( buf, "subi.w #$%X,d%X", imm.w, reg );
            address += 4;
        }
        if ( size == 2 )
        {
            imm.l = memory->GetLongword( address + 2 );
            if ( decImm )sprintf ( buf, "subi.l #%ld,d%X", imm.l, reg );
            else  sprintf ( buf, "subi.l #$%lX,d%X", imm.l, reg );
            address += 6;
        }
    }

    // #data,(an)
    if ( mode == 2 )
    {
        if ( size == 0 )
```

```
    {
        imm.b = memory->GetByte( address + 3 );
        if( decImm )sprintf( buf, "subi.b #%d,(a%X)", imm.b, reg );
        else sprintf( buf, "subi.b #$%X,(a%X)", imm.b, reg );
        address += 4;
    }
    if( size == 1 )
    {
        imm.w = memory->GetWord( address + 2 );
        if( decImm )sprintf( buf, "subi.w #%d,(a%X)", imm.w, reg );
        else sprintf( buf, "subi.w #$%X,(a%X)", imm.w, reg );
        address += 4;
    }
    if( size == 2 )
    {
        imm.l = memory->GetLongword( address + 2 );
        if( decImm )sprintf( buf, "subi.l #%ld,(a%X)", imm.l, reg );
        else sprintf( buf, "subi.l #$%lX,(a%X)", imm.l, reg );
        address += 6;
    }
}

// #data, (an)+
if( mode == 3 )
{
    if( size == 0 )
    {
        imm.b = memory->GetByte( address + 3 );
        if( decImm )sprintf( buf, "subi.b #%d,(a%X)+", imm.b, reg );
        else sprintf( buf, "subi.b #$%X,(a%X)+", imm.b, reg );
        address += 4;
    }
    if( size == 1 )
    {
        imm.w = memory->GetWord( address + 2 );
        if( decImm )sprintf( buf, "subi.w #%d,(a%X)+", imm.w, reg );
        else sprintf( buf, "subi.w #$%X,(a%X)+", imm.w, reg );
        address += 4;
    }
    if( size == 2 )
    {
        imm.l = memory->GetLongword( address + 2 );
        if( decImm )sprintf( buf, "subi.l #%ld,(a%X)+", imm.l, reg );
        else sprintf( buf, "subi.l #$%lX,(a%X)+", imm.l, reg );
        address += 6;
    }
}

// #data, -(an)
if( mode == 4 )
{
    if( size == 0 )
```

```
    {
        imm.b = memory->GetByte( address + 3 );
        if( decImm )sprintf( buf, "subi.b #%d,-(a%X)", imm.b, reg );
        else sprintf( buf, "subi.b #$%X,-(a%X)", imm.b, reg );
        address += 4;
    }
    if( size == 1 )
    {
        imm.w = memory->GetWord( address + 2 );
        if( decImm )sprintf( buf, "subi.w #%d,-(a%X)", imm.w, reg );
        else sprintf( buf, "subi.w #$%X,-(a%X)", imm.w, reg );
        address += 4;
    }
    if( size == 2 )
    {
        imm.l = memory->GetLongword( address + 2 );
        if( decImm )sprintf( buf, "subi.l #%ld,-(a%X)", imm.l, reg );
        else sprintf( buf, "subi.l #$%lX,-(a%X)", imm.l, reg );
        address += 6;
    }
}

// #data, d16(an)
if( mode == 5 )
{
    if( size == 0 )
    {
        displ = (signed short int)memory->GetWord( address + 4 );
        imm.b = memory->GetByte( address + 3 );
        if( decImm )sprintf( buf, "subi.b #%d,%d(a%X)", imm.b, displ, reg );      else sprintf( buf, "subi
            .b #$%X,$%X(a%X)", imm.b, displ, reg );
        address += 6;
    }
    if( size == 1 )
    {
        displ = (signed short int)memory->GetWord( address + 4 );
        imm.w = memory->GetWord( address + 2 );
        if( decImm )sprintf( buf, "subi.w #%d,%d(a%X)", imm.w, displ, reg );      else sprintf( buf, "subi
            .w #$%X,$%X(a%X)", imm.w, displ, reg );
        address += 6;
    }
    if( size == 2 )
    {
        displ = (signed short int)memory->GetWord( address + 6 );
        imm.l = memory->GetLongword( address + 2 );
        if( decImm )sprintf( buf, "subi.l #%ld,%d(a%X)", imm.l, displ, reg );      else sprintf( buf, "subi
            .l #$%lX,$%X(a%X)", imm.l, displ, reg );
        address += 8;
    }
}

// #data, d8(an,xn)
```

```
if ( mode == 6 )
{
   xn = GETBITS( address + 4, 0xF000, 12 );
   if ( size == 0 )
   {
      displ = (signed char)memory->GetByte( address + 5 );
      imm.b = memory->GetByte( address + 3 );
      if ( decImm )sprintf ( buf, "subi.b #%d,%d(a%X,d%X)", imm.b, displ, reg, xn );    else sprintf ( buf
          , "subi.b #$%X,$%X(a%X,d%X)", imm.b, displ, reg, xn );
      address += 6;
   }
   if ( size == 1 )
   {
      displ = (signed char)memory->GetByte( address + 5 );
      imm.w = memory->GetWord( address + 2 );
      if ( decImm )sprintf ( buf, "subi.w #%d,%d(a%X,d%X)", imm.w, displ, reg, xn );    else sprintf ( buf
          , "subi.w #$%X,$%X(a%X,d%X)", imm.w, displ, reg, xn );
      address += 6;
   }
   if ( size == 2 )
   {
      displ = (signed char)memory->GetByte( address + 7 );
      imm.l = memory->GetLongword( address + 2 );
      if ( decImm )sprintf ( buf, "subi.l #%ld,%d(a%X,d%X)", imm.l, displ, reg, xn );        else sprintf (
          buf, "subi.l #$%lX,$%X(a%X,d%X)", imm.l, displ, reg, xn );
      address += 8;
   }
}

if ( mode == 7 )
{
   // #data, (xxx).w
   if ( reg == 0 )
   {
      if ( size == 0 )
      {
         displ = memory->GetWord( address + 4 );
         imm.b = memory->GetByte( address + 3 );
         if ( decImm )sprintf ( buf, "subi.b #%d,$%X", imm.b, displ );
         else sprintf ( buf, "subi.b #$%X,$%X", imm.b, displ );
         address += 6;
      }
      if ( size == 1 )
      {
         displ = memory->GetWord( address + 4 );
         imm.w = memory->GetWord( address + 2 );
         if ( decImm )sprintf ( buf, "subi.w #%d,$%X", imm.w, displ );
         else sprintf ( buf, "subi.w #$%X,$%X", imm.w, displ );
         address += 6;
      }
      if ( size == 2 )
      {
```

```
            displ = memory->GetWord( address + 6 );
            imm.l = memory->GetLongword( address + 2 );
            if( decImm )sprintf( buf, "subi.l #%ld,$%X", imm.l, displ );
            else sprintf( buf, "subi.l #$%lX,$%X", imm.l, displ );
            address += 8;
        }
    }

    // #data, (xxx).l
    if( reg == 1 )
    {
        if( size == 0 )
        {
            displ = memory->GetLongword( address + 4 );
            imm.b = memory->GetByte( address + 3 );
            if( decImm )sprintf( buf, "subi.b #%d,$%X", imm.b, displ );
            else sprintf( buf, "subi.b #$%X,$%X", imm.b, displ );
            address += 8;
        }
        if( size == 1 )
        {
            displ = memory->GetLongword( address + 4 );
            imm.w = memory->GetWord( address + 2 );
            if( decImm )sprintf( buf, "subi.w #%d,$%X", imm.w, displ );
            else sprintf( buf, "subi.w #$%X,$%X", imm.w, displ );
            address += 8;
        }
        if( size == 2 )
        {
            displ = memory->GetLongword( address + 6 );
            imm.l = memory->GetLongword( address + 2 );
            if( decImm )sprintf( buf, "subi.l #%ld,$%X", imm.l, displ );
            else sprintf( buf, "subi.l #$%lX,$%X", imm.l, displ );
            address += 10;
        }
    }
}

    return address;
}

unsigned long M68008::tS_subq( unsigned long address, char *buf )
{
    int size, mode, reg, displ, xn;
    DATA_REGISTER( imm );

    imm.l = GETBITS( address, 0x0E00, 9 );
    size = GETBITS( address, 0x00C0, 6 );
    mode = GETBITS( address, 0x0038, 3 );
    reg = GETBITS( address, 0x0007, 0 );

    // #data,dn
```

```
if ( mode == 0 )
{
    if ( size == 0 )
    {
        if ( decImm )sprintf ( buf, "subq.b␣#%d,d%X", imm.b, reg );
        else sprintf ( buf, "subq.b␣#$%X,d%X", imm.b, reg );
        address += 2;
    }
    if ( size == 1 )
    {
        if ( decImm )sprintf ( buf, "subq.w␣#%d,d%X", imm.b, reg );
        else sprintf ( buf, "subq.w␣#$%X,d%X", imm.b, reg );
        address += 2;
    }
    if ( size == 2 )
    {
        if ( decImm )sprintf ( buf, "subq.w␣#%d,d%X", imm.b, reg );
        else sprintf ( buf, "subq.w␣#$%X,d%X", imm.b, reg );
        address += 2;
    }
}

// #data,an
if ( mode == 1 )
{
    if ( size == 1 )
    {
        if ( decImm )sprintf ( buf, "subq.w␣#%d,a%X", imm.b, reg );
        else sprintf ( buf, "subq.w␣#$%X,a%X", imm.b, reg );
        address += 2;
    }
    if ( size == 2 )
    {
        if ( decImm )sprintf ( buf, "subq.w␣#%d,a%X", imm.b, reg );
        else sprintf ( buf, "subq.w␣#$%X,a%X", imm.b, reg );
        address += 2;
    }
}

// #data,(an)
if ( mode == 2 )
{
    if ( size == 0 )
    {
        if ( decImm )sprintf ( buf, "subq.b␣#%d,(a%X)", imm.b, reg );
        else sprintf ( buf, "subq.b␣#$%X,(a%X)", imm.b, reg );
        address += 2;
    }
    if ( size == 1 )
    {
        if ( decImm )sprintf ( buf, "subq.l␣#%d,(a%X)", imm.b, reg );
        else sprintf ( buf, "subq.l␣#$%X,(a%X)", imm.b, reg );
```

```
        address += 2;
    }
    if( size == 2 )
    {
        if( decImm )sprintf( buf, "subq.w␣#%d,(a%X)", imm.b, reg );
        else sprintf( buf, "subq.w␣#$%X,(a%X)", imm.b, reg );
        address += 2;
    }
}

// #data, (an)+
if( mode == 3 )
{
    if( size == 0 )
    {
        if( decImm )sprintf( buf, "subq.b␣#%d,(a%X)+", imm.b, reg );
        else sprintf( buf, "subq.b␣#$%X,(a%X)+", imm.b, reg );
        address += 2;
    }
    if( size == 1 )
    {
        if( decImm )sprintf( buf, "subq.w␣#%d,(a%X)+", imm.b, reg );
        else sprintf( buf, "subq.w␣#$%X,(a%X)+", imm.b, reg );
        address += 2;
    }
    if( size == 2 )
    {
        if( decImm )sprintf( buf, "subq.l␣#%d,(a%X)+", imm.b, reg );
        else sprintf( buf, "subq.l␣#$%X,(a%X)+", imm.b, reg );
        address += 2;
    }
}

// #data, -(an)
if( mode == 4 )
{
    if( size == 0 )
    {
        if( decImm )sprintf( buf, "subq.b␣#%d,-(a%X)", imm.b, reg );
        else sprintf( buf, "subq.b␣#$%X,-(a%X)", imm.b, reg );
        address += 2;
    }
    if( size == 1 )
    {
        if( decImm )sprintf( buf, "subq.w␣#%d,-(a%X)", imm.b, reg );
        else sprintf( buf, "subq.w␣#$%X,-(a%X)", imm.b, reg );
        address += 2;
    }
    if( size == 2 )
    {
        if( decImm )sprintf( buf, "subq.l␣#%d,-(a%X)", imm.b, reg );
        else sprintf( buf, "subq.l␣#$%X,-(a%X)", imm.b, reg );
```

```
        address += 2;
    }
}

// #data, d16(an)
if( mode == 5 )
{
    displ = (signed short int)memory->GetWord( address + 2 );
    if( size == 0 )
    {
        if( decImm )sprintf( buf, "subq.b #%d,%d(a%X)", imm.b, displ, reg );        else sprintf( buf, "
            subq.b #$%X,$%X(a%X)", imm.b, displ, reg );
        address += 4;
    }
    if( size == 1 )
    {
        if( decImm )sprintf( buf, "subq.w #%d,%d(a%X)", imm.b, displ, reg );        else sprintf( buf, "
            subq.w #$%X,$%X(a%X)", imm.b, displ, reg );
        address += 4;
    }
    if( size == 2 )
    {
        if( decImm )sprintf( buf, "subq.l #%d,%d(a%X)", imm.b, displ, reg );        else sprintf( buf, "
            subq.l #$%X,$%X(a%X)", imm.b, displ, reg );
        address += 4;
    }
}

// #data, d8(an,xn)
if( mode == 6 )
{
    displ = (signed char)memory->GetByte( address + 3 );
    xn = GETBITS( address + 2, 0xF000, 12 );
    if( size == 0 )
    {
        if( decImm )sprintf( buf, "subq.b #%d,%d(a%X,d%X)", imm.b, displ, reg, xn );    else sprintf( buf
            , "subq.b #$%X,$%X(a%X,d%X)", imm.b, displ, reg, xn );
        address += 4;
    }
    if( size == 1 )
    {
        if( decImm )sprintf( buf, "subq.w #%d,%d(a%X,d%X)", imm.b, displ, reg, xn );    else sprintf( buf
            , "subq.w #$%X,$%X(a%X,d%X)", imm.b, displ, reg, xn );
        address += 4;
    }
    if( size == 2 )
    {
        if( decImm )sprintf( buf, "subq.l #%d,%d(a%X,d%X)", imm.b, displ, reg, xn );    else sprintf( buf
            , "subq.l #$%X,$%X(a%X,d%X)", imm.b, displ, reg, xn );
        address += 4;
    }
}
```

```
    if( mode == 7 )
    {
        // #data, (xxx).w
        if( reg == 0 )
        {
            displ = memory->GetWord( address + 2 );
            if( size == 0 )
            {
                if( decImm )sprintf( buf, "subq.b #%d,$%X", imm.b, displ );
                else sprintf( buf, "subq.b #$%X,$%X", imm.b, displ );
            }
            if( size == 1 )
            {
                if( decImm )sprintf( buf, "subq.w #%d,$%X", imm.b, displ );
                else sprintf( buf, "subq.w #$%X,$%X", imm.b, displ );
            }
            if( size == 2 )
            {
                if( decImm )sprintf( buf, "subq.l #%d,$%X", imm.b, displ );
                else sprintf( buf, "subq.l #$%X,$%X", imm.b, displ );
            }
            address += 4;
        }

        // #data, (xxx).l
        if( reg == 1 )
        {
            displ = memory->GetLongword( address + 2 );
            if( size == 0 )
            {
                if( decImm )sprintf( buf, "subq.b #%d,$%X", imm.b, displ );
                else sprintf( buf, "subq.b #$%X,$%X", imm.b, displ );
            }
            if( size == 1 )
            {
                if( decImm )sprintf( buf, "subq.w #%d,$%X", imm.b, displ );
                else sprintf( buf, "subq.w #$%X,$%X", imm.b, displ );
            }
            if( size == 2 )
            {
                if( decImm )sprintf( buf, "subq.l #%d,$%X", imm.b, displ );
                else sprintf( buf, "subq.l #$%X,$%X", imm.b, displ );
            }
            address += 6;
        }
    }

    return address;
}

unsigned long M68008::tS_subx( unsigned long address, char *buf )
```

```
{
    int regY, size , rm, regX;

    regY = GETBITS( address, 0x0E00, 9 );
    size = GETBITS( address, 0x00C0, 6 );
    rm = GETBITS( address, 0x0008, 3 );
    regX = GETBITS( address, 0x0007, 0 );

    if( size == 3 ) // Instruction is in fact SUBA
    {
        return tS_suba( address, buf );
    }

    // subx dx,dy
    if( rm == 0 )
    {
        if( size == 0 )
        {
            sprintf( buf, "subx.b d%X,d%X", regX, regY );
        }
        if( size == 1 )
        {
            sprintf( buf, "subx.w d%X,d%X", regX, regY );
        }
        if( size == 2 )
        {
            sprintf( buf, "subx.l d%X,d%X", regX, regY );
        }
        address += 2;
    }

    // subx -(ax),-(ay)
    if( rm == 1 )
    {
        if( size == 0 )
        {
            sprintf( buf, "subx.b -(a%X),-(a%X)", regX, regY );
        }
        if( size == 1 )
        {
            sprintf( buf, "subx.w -(a%X),-(a%X)", regX, regY );
        }
        if( size == 2 )
        {
            sprintf( buf, "subx.l -(a%X),-(a%X)", regX, regY );
        }
        address += 2;
    }

    return address;
}
```

```
unsigned long M68008::tS_swap( unsigned long address, char *buf )
{
    int reg;

    reg = GETBITS( address, 7, 0 );

    sprintf ( buf, "swap.w␣d%X", reg );
    address += 2;

    return address;
}

unsigned long M68008::tS_tas( unsigned long address, char *buf )
{
    int mode, reg, xn;
    unsigned long displ;

    mode = GETBITS( address, 0x0038, 3 );
    reg  = GETBITS( address, 0x0007, 0 );

    // dn
    if ( mode == 0 )
    {
        sprintf ( buf, "tas.b␣d%X", reg );
        address += 2;
    }

    // (an)
    if ( mode == 2 )
    {
        sprintf ( buf, "tas.b␣(a%X)", reg );
        address += 2;
    }

    // (an)+
    if ( mode == 3 )
    {
        sprintf ( buf, "tas.b␣(a%X)+", reg );
        address += 2;
    }

    // -(an)
    if ( mode == 4 )
    {
        sprintf ( buf, "tas.b␣-(a%X)", reg );
        address += 2;
    }

    // d16(an)
    if ( mode == 5 )
    {
        displ = memory->GetWord( address + 2 );
```

```
        if ( decImm )sprintf( buf, "tas.b␣%ld(a%X)", displ, reg );
        else sprintf( buf, "tas.b␣$%lX(a%X)", displ, reg );
        address += 4;
    }

    // d8(an,xn)
    if ( mode == 6 )
    {
        displ = memory->GetByte( address + 3 );
        xn = GETBITS( address + 2, 0xF000, 12 );
        if ( decImm )sprintf( buf, "tas.b␣%ld(a%X,d%X)", displ, reg, xn );
        else sprintf( buf, "tas.b␣$%lX(a%X,d%X)", displ, reg, xn );
        address += 4;
    }

    if ( mode == 7 )
    {
        // (xxx).w
        if ( reg == 0 )
        {
            displ = memory->GetWord( address + 2 );
            address += 4;
        }

        // (xxx).l
        if ( reg == 1 )
        {
            displ = memory->GetLongword( address + 2 );
            address += 6;
        }

        sprintf( buf, "tas.b␣$%lX", displ );
    }

    return address;
}

unsigned long M68008::tS_trap( unsigned long address, char *buf )
{
    int vector;

    vector = GETBITS( address, 0x000f, 0 );

    if ( decImm )sprintf( buf, "trap␣#%d", vector );
    else sprintf( buf, "trap␣#$%X", vector );

    address += 2;
    return address;
}

unsigned long M68008::tS_trapv( unsigned long address, char *buf )
{
```

```
    sprintf ( buf, "trapv" );

    address += 2;
    return address;
}

unsigned long M68008::tS_tst( unsigned long address, char *buf )
{
    int  size , mode, reg, xn;
    unsigned long displ;

    size = GETBITS( address, 0x00C0, 6 );
    mode = GETBITS( address, 0x0038, 3 );
    reg  = GETBITS( address, 0x0007, 0 );

    // dn
    if ( mode == 0 )
    {
        // byte
        if ( size == 0 )
        {
            sprintf ( buf, "tst.b Dd%X", reg );
        }

        // word
        if ( size == 1 )
        {
            sprintf ( buf, "tst.w Dd%X", reg );
        }

        // long
        if ( size == 2 )
        {
            sprintf ( buf, "tst.l Dd%X", reg );
        }

        address += 2;
    }

    // (an)
    if ( mode == 2 )
    {
        // byte
        if ( size == 0 )
        {
            sprintf ( buf, "tst.b (a%X)", reg );
        }

        // word
        if ( size == 1 )
        {
            sprintf ( buf, "tst.w (a%X)", reg );
```

```
    }

    // long
    if ( size == 2 )
    {
        sprintf ( buf, " tst.l␣(a%X)", reg );
    }

    address += 2;
}

// (an)+
if ( mode == 3 )
{
    // byte
    if ( size == 0 )
    {
        sprintf ( buf, " tst.b␣(a%X)+", reg );
    }

    // word
    if ( size == 1 )
    {
        sprintf ( buf, " tst.w␣(a%X)+", reg );
    }

    // long
    if ( size == 2 )
    {
        sprintf ( buf, " tst.l␣(a%X)+", reg );
    }

    address += 2;
}

// -(an)
if ( mode == 4 )
{
    // byte
    if ( size == 0 )
    {
        sprintf ( buf, " tst.b␣-(a%X)", reg );
    }

    // word
    if ( size == 1 )
    {
        sprintf ( buf, " tst.w␣-(a%X)", reg );
    }

    // long
    if ( size == 2 )
```

```
    {
        sprintf ( buf, " tst.l -(a%X)", reg );
    }

    address += 2;
}

// d16(an)
if ( mode == 5 )
{
    displ = memory->GetWord( address + 2 );

    // byte
    if ( size == 0 )
    {
        if ( decImm )sprintf ( buf, " tst.b %ld(a%X)", displ, reg );
        else  sprintf ( buf, " tst.b $%lX(a%X)", displ, reg );
    }

    // word
    if ( size == 1 )
    {
        if ( decImm )sprintf ( buf, " tst.w %ld(a%X)", displ, reg );
        else  sprintf ( buf, " tst.w $%lX(a%X)", displ, reg );
    }

    // long
    if ( size == 2 )
    {
        if ( decImm )sprintf ( buf, " tst.l %ld(a%X)", displ, reg );
        else  sprintf ( buf, " tst.l $%lX(a%X)", displ, reg );
    }

    address += 4;
}

// d8(an,dn)
if ( mode == 6 )
{
    displ = memory->GetByte( address + 3 );
    xn = GETBITS( address + 2, 0xF000, 12 );

    // byte
    if ( size == 0 )
    {
        if ( decImm )sprintf ( buf, " tst.b %ld(a%X,d%X)", displ, reg, xn );
        else  sprintf ( buf, " tst.b $%lX(a%X,d%X)", displ, reg, xn );
    }

    // word
    if ( size == 1 )
    {
```

```
            if ( decImm )sprintf ( buf, " tst.w␣%ld(a%X,d%X)", displ, reg, xn );
            else  sprintf ( buf, " tst.w␣$%lX(a%X,d%X)", displ, reg, xn );
        }

        // long
        if ( size == 2 )
        {
            if ( decImm )sprintf ( buf, " tst.l␣%ld(a%X,d%X)", displ, reg, xn );
            else  sprintf ( buf, " tst.l␣$%lX(a%X,d%X)", displ, reg, xn );
        }

        address += 4;
    }

    if ( mode == 7 )
    {
        // (xxx).w
        if ( reg == 0 )
        {
            displ = memory->GetWord( address + 2 );

            // byte
            if ( size == 0 )
            {
                sprintf ( buf, " tst.b␣$%lX", displ );
            }

            // word
            if ( size == 1 )
            {
                sprintf ( buf, " tst.w␣$%lX", displ );
            }

            // long
            if ( size == 2 )
            {
                sprintf ( buf, " tst.l␣$%lX", displ );
            }

            address += 4;
        }

        // (xxx).l
        if ( reg == 1 )
        {
            displ = memory->GetLongword( address + 2 );

            // byte
            if ( size == 0 )
            {
                sprintf ( buf, " tst.b␣$%lX", displ );
            }
```

```
            // word
            if ( size == 1 )
            {
                sprintf ( buf, " tst.w $%lX", displ );
            }

            // long
            if ( size == 2 )
            {
                sprintf ( buf, " tst.l $%lX", displ );
            }

            address += 6;
        }
    }

    return address;
}

unsigned long M68008::tS_unlk( unsigned long address, char *buf )
{
    int reg;

    reg = GETBITS( address, 0x0007, 0 );

    sprintf ( buf, " unlk a%X", reg );

    address += 2;
    return address;
}
```

1.10 MemDevice.h

The header file describing the Memory Device class.

```
/*
    Motorola 68008 Simulator
    (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
    Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

    This class represents the memory
*/

#ifndef MEMDEVICE_H
#define MEMDEVICE_H

#include <stdio.h>

class MemDevice
{
public:
```

```
    unsigned char *data;
    unsigned long size;

    MemDevice( unsigned long size );
    ~MemDevice();
    void Clear();
    void Copy( MemDevice *mD );

    unsigned char GetByte( unsigned long address );
    unsigned short int GetWord( unsigned long address );
    unsigned long GetLongword( unsigned long address );

    void SetByte( unsigned char byte, unsigned long address );
    void SetWord( unsigned short int word, unsigned long address );
    void SetLongword( unsigned long longword, unsigned long address );

    bool LoadSRecords( FILE *, unsigned long int * );

private:
    char SRec2Int( char *, unsigned char* );
    unsigned char CharToInt( char );
};

#endif
```

1.11 MemDevice.cpp

The class methods of the Memory Device class (include S-record parsing).

```
/*
    Motorola 68008 Simulator
    (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
    Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

    This class represents the memory
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "MemDevice.h"
#include "M68008.h"

// Constructor
MemDevice::MemDevice( unsigned long _size )
{
    size = _size;
    data = new unsigned char[ size ];

    // Check if memory was actually allocated
    if ( !data )
    {
```

```
        printf( "MemDevice::MemDevice Fatal Error: Out of Memory\n" );
        exit( -1 );
    }

    Clear();
}

// Clear the memory
void MemDevice::Clear()
{
    memset( data, 0, size );
}

// Copy the memory contents of another memdevice into the current
void MemDevice::Copy( MemDevice *mD )
{
    memcpy( data, mD->data, size );
}

// Destructor
MemDevice::~MemDevice()
{
    delete[] data;
}

// Return a byte from memory
unsigned char MemDevice::GetByte( unsigned long address )
{
    static unsigned long pc = 0xFFFFFFFF;
    char *buf;

    // Help stack updates
    if( address > ( size - 1 ) )
    {
        if( m68008->pc != pc )
        {
            buf = new char[ 128 ];
            sprintf( buf, "Bus error at address $%lX: You are trying\n to access a byte outside the memory
                range.", m68008->pc );
            helpStack.push( buf );
            pc = m68008->pc;
        }
        return 0;
    }

    return data[ address ];
}

// Return a word from memory
unsigned short int MemDevice::GetWord( unsigned long address )
{
    static unsigned long pc = 0xFFFFFFFF;
```

```
    char *buf;

    // Help stack updates
    if ( address > ( size - 2 ) )
    {
        if ( m68008->pc != pc )
        {
            buf = new char[ 128 ];
            sprintf ( buf, "Bus error at address $%lX: You are trying\n to access a byte outside the memory
                range.", m68008->pc );
            helpStack.push( buf );
            pc = m68008->pc;
        }
        return 0;
    }
    if ( address & 1 )
    {
        if ( m68008->pc != pc )
        {
            buf = new char[ 128 ];
            sprintf ( buf, "Address error at addres $%lX: The M68k cannot access operands\n longer than one
                byte from an odd memory location.", m68008->pc );
            helpStack.push( buf );
            pc = m68008->pc;
        }
    }

    return ( data[ address ] << 8 ) + data[ address + 1 ];
}

// Return a longword from memory
unsigned long MemDevice::GetLongword( unsigned long address )
{
    static unsigned long pc = 0xFFFFFFFF;
    char *buf;

    // Help stack updates
    if ( address > ( size - 4 ) )
    {
        if ( m68008->pc != pc )
        {
            buf = new char[ 128 ];
            sprintf ( buf, "Bus error at address $%lX: You are trying\n to access a byte outside the memory
                range.", m68008->pc );
            helpStack.push( buf );
            pc = m68008->pc;
        }
        return 0;
    }
    if ( address & 1 )
    {
        if ( m68008->pc != pc )
```

```
        {
            buf = new char[ 128 ];
            sprintf ( buf, "Address error at addres $%lX: The M68k cannot access operands\n longer than one
                byte from an odd memory location.", m68008->pc );
            helpStack.push( buf );
            pc = m68008->pc;
        }
    }

    return ( data[ address ] << 24 ) + ( data[ address + 1 ] << 16 ) +
        ( data[ address + 2 ] << 8 ) + data[ address + 3 ];
}

// Set a byte in memory
void MemDevice::SetByte( unsigned char byte, unsigned long address )
{
    static unsigned long pc = 0xFFFFFFFF;
    char *buf;

    // Help stack updates
    if ( address > ( size - 1 ) )
    {
        if ( m68008->pc != pc )
        {
            buf = new char[ 128 ];
            sprintf ( buf, "Bus error at address $%lX: You are trying\n to access a byte outside the memory
                range.", m68008->pc );
            helpStack.push( buf );
            pc = m68008->pc;
        }
        return;
    }
    data[ address ] = byte;
}

// Set a word in memory
void MemDevice::SetWord( unsigned short int word, unsigned long address )
{
    static unsigned long pc = 0xFFFFFFFF;
    char *buf;

    // Help stack updates
    if ( address > ( size - 2 ) )
    {
        if ( m68008->pc != pc )
        {
            buf = new char[ 128 ];
            sprintf ( buf, "Bus error at address $%lX: You are trying\n to access a byte outside the memory
                range.", m68008->pc );
            helpStack.push( buf );
            pc = m68008->pc;
        }
```

```
        return;
    }
    if ( address & 1 )
    {
        if ( m68008->pc != pc )
        {
            buf = new char[ 128 ];
            sprintf ( buf, "Address error at addres $%lX: The M68k cannot access operands\n longer than one
                byte from an odd memory location.", m68008->pc );
            helpStack.push( buf );
            pc = m68008->pc;
        }
    }

    data[ address + 0 ] = (unsigned char)( ( word & 0xFF00 )>> 8 );
    data[ address + 1 ] = (unsigned char)( ( word & 0x00FF ));
}

// Set a longword in memory
void MemDevice::SetLongword( unsigned long longword, unsigned long address )
{
    static unsigned long pc = 0xFFFFFFFF;
    char *buf;

    // Help stack updates
    if ( address > ( size - 4 ) )
    {
        if ( m68008->pc != pc )
        {
            buf = new char[ 128 ];
            sprintf ( buf, "Bus error at address $%lX: You are trying\n to access a byte outside the memory
                range.", m68008->pc );
            helpStack.push( buf );
            pc = m68008->pc;
        }
        return;
    }
    if ( address & 1 )
    {
        if ( m68008->pc != pc )
        {
            buf = new char[ 128 ];
            sprintf ( buf, "Address error at addres $%lX: The M68k cannot access operands\n longer than one
                byte from an odd memory location.", m68008->pc );
            helpStack.push( buf );
            pc = m68008->pc;
        }
    }

    data[ address + 0 ] = (unsigned char)( ( longword & 0xFF000000 )>> 24 );
    data[ address + 1 ] = (unsigned char)( ( longword & 0x00FF0000 )>> 16 );
    data[ address + 2 ] = (unsigned char)( ( longword & 0x0000FF00 )>> 8 );
```

```
    data[ address + 3 ] = (unsigned char)( ( longword & 0x000000FF ));
}

// Load S-Record file into memory
// Returns TRUE on success, FALSE on failure
// Sets initPC to the (probable) starting point of the application
bool MemDevice::LoadSRecords( FILE *in, unsigned long int *initPC )
{
    char ascBuffer[ 256 ];          // ASCII buffer
    unsigned char intBuffer[ 128 ]; // Integer buffer
    char numCharPairs;              // Number of character pairs in one S-record
    unsigned long address;          // The address specified in the S-record
    int lp1;
    bool addrSet = false;

    *initPC = 0;

    for ( ;; )
    {
        fscanf( in, "%s", ascBuffer );
        if( feof( in ) ) break;

        // Empty line?
        if ( !ascBuffer[ 0 ] )
            continue;

        // Convert to integer values
        numCharPairs = SRec2Int( ascBuffer, intBuffer );
        if( numCharPairs == -1 )
            return false;

        // Now it depends on the type of S-record what we need to do
        switch( intBuffer[ 0 ] )
        {
            case 0:
                // The type-0 S-record contains information on the name
                // of the module etc. We'll ignore it for the moment
                break;
            case 1:
                // Type-1 S-records contain memory loadable data, with a 2-byte
                // address, finished with a checksum (which we ignore)
                address = ( intBuffer[ 2 ] << 8 ) + intBuffer[ 3 ];
                if ( !addrSet )
                {
                    *initPC = address;
                    addrSet = true;
                }
                for( lp1 = 0; lp1 < intBuffer[ 1 ] - 3; lp1++ )
                    data[ address++ ] = intBuffer[ lp1 + 4 ];
                break;
            case 2:
                // Type-2 S-records contain memory loadable data, with a 3-byte
```

```
    // address, finished with a checksum (which we ignore)
    address = ( intBuffer [ 2 ] << 16 ) + ( intBuffer [ 3 ] << 8 ) + intBuffer [ 4 ];
    if ( !addrSet )
    {
        *initPC = address;
        addrSet = true;
    }
    for( lp1 = 0; lp1 < intBuffer [ 1 ] - 4; lp1++ )
        data[ address++ ] = intBuffer[ lp1 + 5 ];
    break;
case 3:
    // Type-3 S-records contain memory loadable data, with a 4-byte
    // address, finished with a checksum (which we ignore)
    address = ( intBuffer [ 2 ] << 24 ) + ( intBuffer [ 3 ] << 16 ) +
        ( intBuffer [ 4 ] << 8 ) + intBuffer [ 5 ];
    if ( !addrSet )
    {
        *initPC = address;
        addrSet = true;
    }
    for( lp1 = 0; lp1 < intBuffer [ 1 ] - 5; lp1++ )
        data[ address++ ] = intBuffer[ lp1 + 6 ];
    break;
case 4:
    // Type-4 S-records do not exist
    return false;
case 5:
    // Type-5 S-records contains the number of previously submitted S-records
    // We ignore it
    break;
case 6:
    // Type-6 S-records do not exist
    return false;
case 7:
    // Type-7 S-records specify a 4-byte starting execution address
    // If it is set, we make initPC equal to this address
    address = ( intBuffer [ 2 ] << 24 ) + ( intBuffer [ 3 ] << 16 ) +
        ( intBuffer [ 4 ] << 8 ) + intBuffer [ 5 ];
    if ( address )
        *initPC = address;
    break;
case 8:
    // Type-8 S-records specify a 3-byte starting execution address
    // If it is set, we make initPC equal to this address
    address = ( intBuffer [ 2 ] << 16 ) + ( intBuffer [ 3 ] << 8 ) + intBuffer [ 4 ];
    if ( address )
        *initPC = address;
    break;
case 9:
    // Type-9 S-records specify a 2-byte starting execution address
    // If it is set, we make initPC equal to this address
    address = ( intBuffer [ 2 ] << 8 ) + intBuffer [ 3 ];
```

```
            if ( address )
               *initPC = address;
            break;
         }
      }

   return true;
}

// Convert a single S-record from ASCII to integers (private function)
// Returns the number of converted character pairs or -1 on failure
char MemDevice::SRec2Int( char *ascBuffer, unsigned char *intBuffer )
{
   int index = 0;
   unsigned char tc1, tc2;
   char numConv = 1;

   // Check if the S-record starts with a 'S'
   if ( ascBuffer [ 0 ] != 'S' )
      return -1;

   // Store the type of S-record
   intBuffer [ index++ ] = CharToInt( ascBuffer[ 1 ] );

   // Accept known S-record types only
   if ( intBuffer [ 0 ] > 9 )
      return -1;

   // And convert the remaining character pairs one by one
   ascBuffer += 2;  // Skip to first character pair
   while( *ascBuffer )
   {
      tc1 = CharToInt( ascBuffer[ 0 ] );
      tc2 = CharToInt( ascBuffer[ 1 ] );
      if ( ( tc1 == 255 ) || ( tc2 == 255 )) return -1;
      intBuffer [ index++ ] = ( tc1 << 4 )+ tc2;
      ascBuffer += 2;
      numConv++;
   }

   return numConv;
}

// Returns the converted value of a character ('0' - '9' is mapped to 0 - 9
// and 'A' - 'F' and 'a' - 'f' are mapped to 10 - 15), or 255 on failure
unsigned char MemDevice::CharToInt( char c )
{
   if ( ( c >= '0' ) && ( c <= '9' ) ) return c - '0';
   if ( ( c >= 'A' ) && ( c <= 'F' )) return c - 'A' + 10;
   if ( ( c >= 'a' ) && ( c <= 'f' ) ) return c - 'a' + 10;
   return 255;
}
```

1.12 stack.h

Generic stack class definition.

```
/*
   Motorola 68008 Simulator
   (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
   Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

   This class represents a generic stack
*/

template<class T>
class Node
{
public:
   Node<T> *next, *prev;
   T item;
   Node( T i );
};

template<class T>
class Stack
{
private:
   Node<T> *top;
   int count;
public:
   Stack();
   ~Stack();
   void reset();
   void push( T item );
   T pop();
   T getTop();
   void setTop( T item );
   T getItem( int nr );
   int getCount();
};
```

1.13 stack.cpp

Generic stack class members.

```
/*
   Motorola 68008 Simulator
   (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
   Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

   This class represents a generic stack
*/

#include "stack.h"
```

```
template<class T>
Node<T>::Node( T i )
{
   item = i;
   prev = next = NULL;
}

template<class T>
Stack<T>::Stack()
{
   top = NULL;
   count = 0;
}

template<class T>
Stack<T>::~Stack()
{
   // to be filled in later
}

template<class T>
void Stack<T>::reset()
{
   while ( count > 0 )
   {
      pop();
      count--;
   }
}

template<class T>
void Stack<T>::push( T item )
{
   Node<T> *newNode = new Node<T>( item );

   newNode->prev = top;
   if ( top )
      top->next = newNode;
   top = newNode;

   count++;
}

template<class T>
T Stack<T>::pop()
{
   T Result;

   count--;

   // This is a bad solution (exception handling would be better)
```

```
    // However, that's too complicated :-) Just make sure you check if you can pop off the stack first
    // using the count() method
    if ( !top )
        return 0;

    Result = top->item;

    if( top->prev )
    {
        top = top->prev;
        delete top->next;
        top->next = NULL;
    }
    else
    {
        delete top;
        top = NULL;
    }

    return Result;
}

template<class T>
T Stack<T>::getTop()
{
    return top->item;
}

template<class T>
void Stack<T>::setTop( T item )
{
    top->item = item;
}

// Return the nr's item from the top
template<class T>
T Stack<T>::getItem( int nr )
{
    int lp1;
    Node<T> *current = top;

    for( lp1 = 0; lp1 < nr; lp1++ )
    {
        current = current->prev;
    }

    return current->item;
}

template<class T>
int Stack<T>::getCount()
{
```

```
    return count;
}
```

1.14 stackItem.h

The stackItem class is used in combination with the generic stack class. It is used to store information about the system stack viewed at a logical level (this is the counterpart of the system stack display in the GUI).

```
/*
    Motorola 68008 Simulator
    (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
    Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

    System Stack item class.
*/


#ifndef STACKITEM_H
#define STACKITEM_H

#include <qstring.h>

class StackItem{
public:
    StackItem();
    StackItem( short int _ID, unsigned long int _address, char *_origin );
    short int ID;
    unsigned long int address;
    char origin [ 20 ];
};

#endif
```

1.15 stackItem.cpp

And the stackItem member functions.

```
/*
    Motorola 68008 Simulator
    (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
    Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

    System Stack item class.
*/

#include "stackItem.h"
#include "M68008.h"

StackItem::StackItem()
{
    ID = 0;
```

```
    address = 0;
    memset( origin , 0, 20 ) ;
}

StackItem::StackItem( short int _ID, unsigned long int _address, char *_origin )
{
    ID = _ID;
    address = _address;
    strcpy( origin , _origin ) ;
}
```

Part II

Frontend (GUI)

1.16 main.cpp

This is the main application file for the program in graphical mode (see also 68sim.cpp in part I). This, and any of the other files in the GUI section, should *not* be compiled when the program should be compiled for console mode.

```
/*
   Motorola 68008 Simulator
   (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
   Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

   This is the main application file for the GUI
*/

#include <qapplication.h>
#include <qmessagebox.h>

#include "gui.h"
#include "../M68008.h"

M68008 *m68008;
MemDevice *memory, *bMemory;

Stack<char*> helpStack;          // Contains the help system messages
Stack<unsigned long int> pushPop; // Stores the SSP per subroutine
Stack<bool> pushStack;           // Checks if a subroutine uses the stack at all

Stack<StackItem*> systemStack;   // Contains the items displayed in stackTable
int maxID = 0;

int main( int argc, char **argv, char **env )
{
   /* Initialize the backend */

   // Initialize memory (1 MB or 0x100000 bytes)
   memory = new MemDevice( 1048576 );
   bMemory = new MemDevice( 1048576 );

   // Initialize the CPU
   m68008 = new M68008( memory, bMemory, 0x100000 );

   /* And initialize the GUI */

   QApplication app( argc, argv );

   // Should add Minimise, Maximise and ContextHelp buttons but only adds Minimise and Maximise.
   //gui dialog ( 0, 0, TRUE, Qt::WStyle_MinMax | Qt::WStyle_ContextHelp);
   gui dialog ( 0, 0, TRUE, Qt::WStyle_Customize | Qt::WStyle_NormalBorder | Qt::WStyle_Title |
      Qt::WStyle_SysMenu | Qt::WStyle_Minimize | Qt::WStyle_Maximize | Qt::WStyle_ContextHelp | Qt::
         WType_TopLevel );
   app.setMainWidget(&dialog);

   dialog.exec();
   return 0;
```

}

1.17 guiBase.h

The base definitions for the GUI. The actual GUI window is defined in gui.h and inherits from this class.

```
/*
   Motorola 68008 Simulator
   (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
   Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

   The GUI elements
*/

#ifndef SKELETONCODEDIALOGBASE_H
#define SKELETONCODEDIALOGBASE_H

#include <qvariant.h>
#include <qdialog.h>
#include <qlabel.h>
#include <qimage.h>
#include <qtable.h>
#include <qpixmap.h>
#include <qtimer.h>
#include <qcheckbox.h>

class QVBoxLayout;
class QHBoxLayout;
class QGridLayout;
class QLCDNumber;
class QPushButton;
class QSpinBox;

class QMTable : public QTable
{
   Q_OBJECT

protected:
   void keyPressEvent( QKeyEvent* e );
   void keyReleaseEvent( QKeyEvent* e );

public:
   QMTable( QWidget * parent = 0, const char * name = 0, int cols = 0 )
      : QTable( parent, name )
   {
      int lp1 = 0;

      align = new int[ cols ];
      for( lp1 = 0; lp1 < cols; lp1++ )
         align [ lp1 ] = Qt::AlignRight;

      hlRow = hlCol = -1;
```

```
        hlRows = false;

        hlRow2 = hlCol2 = -1;
        hlRows2 = false;
        allCols = cols;

        CtrlPressed = false;
    }

    void setColumnsStretchable ( int start, bool stretch );
    void paintCell( QPainter * p, int row, int col, const QRect & cr, bool selected );

    int *align;
    int hlRow, hlCol;     // Row and column to be highlighted (-1 for none)
    bool hlRows;          // Highlight rows rather than individual cells?
    bool CtrlPressed;

    int hlRow2, hlCol2;
    int allCols;
    bool hlRows2;
};

class guiBase : public QDialog
{
    Q_OBJECT

public:
    guiBase( QWidget* parent = 0, const char* name = 0, bool modal = FALSE, WFlags fl = 0 );
    ~guiBase();

    QMTable* registers;
    QMTable* stackTable;
    QMTable* helpStackTable;
    QMTable* memoryViewer;
    QMTable* otherRegisters;
    QMTable* programViewer;
    QSpinBox* adjustSpeed;
    QLCDNumber* speed;
    QLabel* opcodeDefinition;
    QPushButton* open;
    QPushButton* restart;
    QPushButton* step;
    QPushButton* undo;
    QPushButton* run;
    QPushButton* go;
    QPushButton *ledZ, *ledV, *ledN, *ledX, *ledC;
    QTimer *timer;
    QPixmap* opcodeBitmap;
    QCheckBox* decHex;
    QCheckBox* autoScroll;
};
```

#endif *// SKELETONCODEDIALOGBASE_H*

1.18 guiBase.cpp

And the member functions of the GUI base.

```
/*
   Motorola 68008 Simulator
   (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
   Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

   Graphical User Interface - Set up of the GUI elements
*/

#include "guiBase.h"

#include <stdio.h>

#include <qheader.h>
#include <qlabel.h>
#include <qlcdnumber.h>
#include <qpushbutton.h>
#include <qspinbox.h>
#include <qtable.h>
#include <qlayout.h>
#include <qvariant.h>
#include <qtooltip.h>
#include <qwhatsthis.h>
#include <qtable.h>
#include <qmessagebox.h>
#include <qlayout.h>

guiBase::guiBase( QWidget* parent, const char* name, bool modal, WFlags fl )
    : QDialog( parent, name, modal, fl )
{
    QHeader *header;
    int lp1; char buf[20];
    QString str;

    // Main Window
    setName( "Motorola_68008_Simulator" );
    setProperty( "caption", tr( "Motorola_68008_Simulator" ));

    // Layout
    QGridLayout *window = new QGridLayout( this, 1, 1 );
    window->setMargin( 2 );
    QHBoxLayout *top = new QHBoxLayout( window );
    top->setMargin( 2 );
    QHBoxLayout *bottom = new QHBoxLayout( window );
    bottom->setMargin( 2 );
    QVBoxLayout *leftBottom = new QVBoxLayout( bottom );
    leftBottom->setMargin( 2 );
```

```
    leftBottom->setSpacing( 2 );
    QVBoxLayout *rightBottom = new QVBoxLayout( bottom );
    rightBottom->setMargin( 2 );
    rightBottom->setSpacing( 2 );

    // Buttons
    QHBoxLayout *buttons = new QHBoxLayout( top );
    buttons->setSpacing( 4 );

    open = new QPushButton( this, "open" );
    open->setProperty( "text", tr( "Open" ));
// open->setMinimumSize( 80, 30 );
    buttons->addWidget( open );

    restart = new QPushButton( this, "restart" );
    restart->setProperty( "text", tr( "Reset" ) );
// restart->setMinimumSize( 80, 30 );
    buttons->addWidget( restart );

    step = new QPushButton( this, "step" );
    step->setProperty( "text", tr( "Step" ) );
// step->setMinimumSize( 80, 30 );
    buttons->addWidget( step );

    undo = new QPushButton( this, "undo" );
    undo->setProperty( "text", tr( "Undo" ));
// undo->setMinimumSize( 80, 30 );
    buttons->addWidget( undo );

    run = new QPushButton( this, "run" );
    run->setProperty( "text", tr( "Run" ));
// run->setMinimumSize( 80, 30 );
    buttons->addWidget( run );

    go = new QPushButton( this, "go" );
    go->setProperty( "text", tr( "Go!" ) );
// go->setMinimumSize( 80, 30 );
    buttons->addWidget( go );

    top->addStretch();

    // Program Options
    QVBoxLayout* options;
    options = new QVBoxLayout( top );
    options->setSpacing( 4 );

    autoScroll = new QCheckBox( this, "autoScroll" );
    autoScroll->setText( "Auto-scroll Program Viewer" );
    autoScroll->setChecked( TRUE );
    options->addWidget( autoScroll );

    decHex = new QCheckBox( this, "decHex" );
```

```
    decHex->setText( "Show constants in hex" );
    decHex->setChecked( TRUE );
    options->addWidget( decHex );

    // The run speed selector
    adjustSpeed = new QSpinBox( 50, 5000, 50, this, "adjustSpeed" );
    adjustSpeed->setValue( 500 );
    adjustSpeed->setMinimumSize( 62, 32 );
    top->addWidget( adjustSpeed );

    // Stack Viewer
    stackTable = new QMTable( this, "stackTable", 3 );
    stackTable->setProperty( "numRows", 128 );
    stackTable->setProperty( "numCols", 3 );
    stackTable->setLeftMargin( 0 );
// stackTable->setMinimumWidth( 290 );
// stackTable->setMinimumSize( 290, 262 );
    stackTable->setColumnsStretchable( 0, true );
    rightBottom->addWidget( stackTable );

    header = stackTable->horizontalHeader();
    header->setLabel( 0, "Address", 90 );
    header->setLabel( 1, "Origin", 90 );
    header->setLabel( 2, "Contents", 90 );
    header->setResizeEnabled( false );

    // Program Viewer
    programViewer = new QMTable( this, "programViewer", 3 );
    programViewer->setProperty( "numRows", 128 );
    programViewer->setProperty( "numCols", 3 );
    programViewer->setLeftMargin( 0 );
    programViewer->align[ 2 ] = Qt::AlignLeft;
    programViewer->setMinimumWidth( 430 );
    //programViewer->setMinimumSize( 450, 294 );
    programViewer->setColumnsStretchable( 0, true );
    leftBottom->addWidget( programViewer );

    header = programViewer->horizontalHeader();
    header->setLabel( 0, "Address", 90 );
    header->setLabel( 1, "Opcode", 150 );
    header->setLabel( 2, "Decoding", 190 );
    header->setResizeEnabled( false );

    for( lp1 = 0; lp1 < 128; lp1++ )
    {
        programViewer->setItem( lp1, 1, new QTableItem( programViewer, QTableItem::Never,
            NULL ));
        programViewer->setItem( lp1, 2, new QTableItem( programViewer, QTableItem::Never,
            NULL ));
    }

    // Help stack
```

```
helpStackTable = new QMTable( this, "helpStackTable", 1 );
helpStackTable->setProperty( "numRows", 128 );
helpStackTable->setProperty( "numCols", 1 );
helpStackTable->setLeftMargin( 30 );
helpStackTable->setTopMargin( 0 );
helpStackTable->align[ 0 ] = Qt::AlignCenter;
helpStackTable->setMinimumHeight( 44 );
helpStackTable->setMaximumHeight( 44 );
helpStackTable->setColumnsStretchable( 0, true );
leftBottom->addWidget( helpStackTable );

header = helpStackTable->horizontalHeader();
header->setLabel( 0, NULL, 400 );
header->setResizeEnabled( false );

header = helpStackTable->verticalHeader();
header->setResizeEnabled( false );
for( lp1 = 0; lp1 < 128; lp1++ )
    header->resizeSection( lp1, 40 );

// Memory Viewer
memoryViewer = new QMTable( this, "memoryViewer", 17 );
memoryViewer->setProperty( "numRows", 128 );
memoryViewer->setProperty( "numCols", 17 );
memoryViewer->setLeftMargin( 0 );
//memoryViewer->setMinimumWidth( 463 );
//memoryViewer->setMinimumSize( 450, 100 );
memoryViewer->setColumnsStretchable( 1, true );
memoryViewer->setColumnsStretchable( 14, false );
leftBottom->addWidget( memoryViewer );

header = memoryViewer->horizontalHeader();
header->setLabel( 0, "Address", 78 );
header->setResizeEnabled( false );
for( lp1 = 0; lp1 < 16; lp1++ )
{
    sprintf ( buf, "%d", lp1 );
    header->setLabel( lp1 + 1, buf, 22 );
}

// The opcode template
opcodeDefinition = new QLabel( this, "opcodeDefinition" );
opcodeDefinition->setBackgroundColor( QColor( 255, 255, 255 ));
 opcodeDefinition->setProperty( "frameShape", (int)QLabel::Panel );
 opcodeDefinition->setProperty( "frameShadow", (int)QLabel::Sunken );
 opcodeDefinition->setProperty( "text", tr( "opcode_bitmap_:)" ));
opcodeDefinition->setMinimumSize( 463, 62 );
rightBottom->addWidget( opcodeDefinition );

// Layout for next region
QHBoxLayout *rightBottomCorner = new QHBoxLayout( rightBottom );
QHBoxLayout *cpuRegs = new QHBoxLayout( rightBottomCorner );
```

```
QVBoxLayout *final = new QVBoxLayout( rightBottomCorner );
QHBoxLayout *otherRegs = new QHBoxLayout( final );

// Registers
 registers  = new QMTable( this, "registers", 2 );
 registers->setProperty( "numRows", 8 );
 registers->setProperty( "numCols", 2 );
 registers->setLeftMargin( 20 );
 registers->setMinimumSize( 204, 183 );
 registers->setMaximumSize( 204, 183 );
 registers->setColumnsStretchable( 0, true );
cpuRegs->addWidget( registers );
cpuRegs->addStretch();

header = registers->horizontalHeader();
header->setLabel( 0, "Data Regs", 90 );
header->setLabel( 1, "Address Regs", 90 );
header->setResizeEnabled( false );
header = registers->verticalHeader();
header->setResizeEnabled( false );
for( lp1 = 0; lp1 < 8; lp1++ )
   header->setLabel( lp1, str.sprintf( "%d", lp1 ) );

// And the "other" registers
otherRegisters = new QMTable( this, "otherRegisters", 1 );
otherRegisters->setProperty( "numRows", 3 );
otherRegisters->setProperty( "numCols", 1 );
otherRegisters->setTopMargin( 0 );
otherRegisters->setItem( 1, 0, new QTableItem( otherRegisters, QTableItem::Never,
      NULL ));
otherRegisters->setItem( 2, 0, new QTableItem( otherRegisters, QTableItem::Never,
      NULL ));
otherRegisters->setMinimumSize( 125, 64 );
otherRegisters->setMaximumSize( 125, 64);
otherRegisters->setColumnsStretchable( 0, true );
otherRegs->addWidget( otherRegisters );
otherRegs->addStretch();

header = otherRegisters->verticalHeader();
header->setResizeEnabled( false );
header->setLabel( 0, "PC" );
header->setLabel( 1, "IR" );
header->setLabel( 2, "CCR" );
header = otherRegisters->horizontalHeader();
header->setResizeEnabled( false );
header->resizeSection( 0, 90 );

// The "LEDs"
QGridLayout *ledDisplay = new QGridLayout( final, 2, 4 );
QLabel *tLabel;

ledX = new QPushButton( this, "ledX" );
```

```
ledX->setPalette( QPalette( QColor( 255, 10, 10 )) );
ledX->setMinimumSize( 16, 8 );
ledX->setMaximumSize( 20, 10 );
ledDisplay->addWidget( ledX, 0, 1 );

tLabel = new QLabel( this, "ledXLab" );
tLabel->setText( "eXtend" );
tLabel->setMinimumSize( 50, 16 );
ledDisplay->addWidget( tLabel, 0, 0 );

ledN = new QPushButton( this, "ledN" );
ledN->setPalette( QPalette( QColor( 255, 10, 10 )) );
ledN->setMinimumSize( 16, 8 );
ledN->setMaximumSize( 20, 10 );
ledDisplay->addWidget( ledN, 1, 1 );

tLabel = new QLabel( this, "ledNLab" );
tLabel->setText( "Negative" );
tLabel->setMinimumSize( 50, 16 );
ledDisplay->addWidget( tLabel, 1, 0 );

ledZ = new QPushButton( this, "ledZ" );
ledZ->setPalette( QPalette( QColor( 255, 10, 10 )) );
ledZ->setMinimumSize( 16, 8 );
ledZ->setMaximumSize( 20, 10 );
ledDisplay->addWidget( ledZ, 2, 1 );

tLabel = new QLabel( this, "ledZLab" );
tLabel->setText( "Zero" );
tLabel->setMinimumSize( 50, 16 );
ledDisplay->addWidget( tLabel, 2, 0 );

ledV = new QPushButton( this, "ledV" );
ledV->setPalette( QPalette( QColor( 255, 10, 10 )) );
ledV->setMinimumSize( 16, 8 );
ledV->setMaximumSize( 20, 10 );
ledDisplay->addWidget( ledV, 3, 1 );

tLabel = new QLabel( this, "ledVLab" );
tLabel->setText( "oVerflow" );
tLabel->setMinimumSize( 50, 16 );
ledDisplay->addWidget( tLabel, 3, 0 );

ledC = new QPushButton( this, "ledC" );
ledC->setPalette( QPalette( QColor( 255, 10, 10 )) );
ledC->setMinimumSize( 16, 8 );
ledC->setMaximumSize( 20, 10 );
ledDisplay->addWidget( ledC, 4, 1 );

tLabel = new QLabel( this, "ledCLab" );
tLabel->setText( "Carry" );
tLabel->setMinimumSize( 50, 16 );
```

```
    ledDisplay->addWidget( tLabel, 4, 0);

    ledDisplay->setRowStretch(5, 0);

    // Set window to minimum size
    // window gets thrown into a corner if minimumWidth(), minimumHeight() are used
    resize ( 754, 565 ) ;

    // Initialize the timer
    timer = new QTimer( this, "run_timer" );

}

/*
 * Destroys the object and frees any allocated resources
 */
guiBase::~guiBase()
{
    // no need to delete child widgets, Qt does it all for us
}
```

1.19 gui.h

The remaining definitions for the GUI (see also guibase.h). The member definitions for this header file are distributed over the remaining GUI files.

```
/*
    Motorola 68008 Simulator
    (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
    Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

    The auxiliary GUI methods
*/

#include "guiBase.h"
#include <qwhatsthis.h>
#include <qstatusbar.h>

class gui : public guiBase
{
    Q_OBJECT
public:
    gui( QWidget* parent = 0, const char* name = 0, bool modal = FALSE, WFlags f = 0 );
    //void openFileFromArgs( int argc, char** argv );

protected:
    void helpStackTableInitialise();
    void memoryViewerInitialise();
    void programViewerInitialise();
    void registersInitialise();
    void stackTableInitialise();
```

```
    void whatsThisInitialise();
    void setOpcodeBitmap( const char* instrName );

    void keyPressEvent( QKeyEvent* e );
    void keyReleaseEvent( QKeyEvent* e);

signals:
    void updateTables();

public slots:
    void memoryViewerCurrentChanged( int row, int col );
    void memoryViewerClicked( int row, int col, int button, const QPoint &mousePos );
    void memoryViewerDoubleClicked( int row, int col, int button, const QPoint &mousePos );
    void memoryViewerPressed( int row, int col, int button, const QPoint &mousePos );
    void memoryViewerSelectionChanged();
    void memoryViewerValueChanged( int row, int col );
    void memoryViewerUpdate();

    void helpStackTableCurrentChanged( int row, int col );
    void helpStackTableClicked( int row, int col, int button, const QPoint &mousePos );
    void helpStackTableDoubleClicked( int row, int col, int button, const QPoint &mousePos );
    void helpStackTablePressed( int row, int col, int button, const QPoint &mousePos );
    void helpStackTableSelectionChanged();
    void helpStackTableValueChanged( int row, int col );
    void helpStackTableUpdate();
    void helpStackTableClearHl();

    void programViewerCurrentChanged( int row, int col );
    void programViewerClicked( int row, int col, int button, const QPoint &mousePos );
    void programViewerDoubleClicked( int row, int col, int button, const QPoint &mousePos );
    void programViewerPressed( int row, int col, int button, const QPoint &mousePos );
    void programViewerSelectionChanged();
    void programViewerValueChanged( int row, int col );
    void programViewerUpdate();
    void programViewerQuickUpdate( char* lastInstr );

    void registersCurrentChanged( int row, int col );
    void registersClicked( int row, int col, int button, const QPoint &mousePos );
    void registersDoubleClicked( int row, int col, int button, const QPoint &mousePos );
    void registersPressed( int row, int col, int button, const QPoint &mousePos );
    void registersSelectionChanged();
    void registersValueChanged( int row, int col );
    void registersUpdate();

    void otherRegistersCurrentChanged( int row, int col );
    void otherRegistersClicked( int row, int col, int button, const QPoint &mousePos );
    void otherRegistersDoubleClicked( int row, int col, int button, const QPoint &mousePos );
    void otherRegistersPressed( int row, int col, int button, const QPoint &mousePos );
    void otherRegistersSelectionChanged();
    void otherRegistersValueChanged( int row, int col );

    void stackTableCurrentChanged( int row, int col );
```

```
    void stackTableClicked( int row, int col, int button, const QPoint &mousePos );
    void stackTableDoubleClicked( int row, int col, int button, const QPoint &mousePos );
    void stackTablePressed( int row, int col, int button, const QPoint &mousePos );
    void stackTableSelectionChanged();
    void stackTableValueChanged( int row, int col );
    void stackTableUpdate();

    void openClicked();
    void restartClicked();
    void stepClicked();
    void undoClicked();
    void runClicked();
    void goClicked();

    void adjustSpeedValueChanged( int value );

    void ledXClicked();
    void ledNClicked();
    void ledZClicked();
    void ledVClicked();
    void ledCClicked();

    void decHexClicked();
    void decHexStateChanged( int state );
    void autoScrollClicked();
    void autoScrollStateChanged( int state );

private:
    bool pvCellEdited;
    bool CtrlPressed;
    unsigned long memBase, progBase, breakPoint;
    QWhatsThis *DialogWhatsThis;
    QStatusBar *DialogStatusBar;
};
```

1.20 gui.cpp

This file mainly sets up signal/slot connections for event handling and sets up the "What's This?" functionality.

```
/*
    Motorola 68008 Simulator
    (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
    Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

    This file sets up the rest of the GUI (see also: guiBase.cpp)
*/

#include "gui.h"
#include <qmessagebox.h>
#include <qpoint.h>

#include "../M68008.h"
```

```
gui::gui( QWidget* parent, const char* name, bool modal, WFlags f )
    : guiBase( parent, name, modal, f )
{
    // Signal connections for memoryViewer
    QObject::connect( (QObject *)memoryViewer, SIGNAL( currentChanged( int, int )),
        this, SLOT( memoryViewerCurrentChanged( int, int )));

    QObject::connect( (QObject *)memoryViewer, SIGNAL( clicked( int, int, int, const QPoint & )),
        this, SLOT( memoryViewerClicked( int, int, int, const QPoint & )));

    QObject::connect( (QObject *)memoryViewer, SIGNAL( doubleClicked( int, int, int, const QPoint & )),
        this, SLOT( memoryViewerDoubleClicked( int, int, int, const QPoint & )));

    QObject::connect( (QObject *)memoryViewer, SIGNAL( pressed( int, int, int, const QPoint & )),
        this, SLOT( memoryViewerPressed( int, int, int, const QPoint & )));

    QObject::connect( (QObject *)memoryViewer, SIGNAL( selectionChanged() ),
        this, SLOT( memoryViewerSelectionChanged() ));

    QObject::connect( (QObject *)memoryViewer, SIGNAL( valueChanged( int, int )),
        this, SLOT( memoryViewerValueChanged( int, int )));

    // signal connections for programViewer
    QObject::connect( (QObject *)programViewer, SIGNAL( currentChanged( int, int )),
        this, SLOT( programViewerCurrentChanged( int, int )));

    QObject::connect( (QObject *)programViewer, SIGNAL( clicked( int, int, int, const QPoint & )),
        this, SLOT( programViewerClicked( int, int, int, const QPoint & )));

    QObject::connect( (QObject *)programViewer, SIGNAL( doubleClicked( int, int, int, const QPoint & )),
        this, SLOT( programViewerDoubleClicked( int, int, int, const QPoint & )));

    QObject::connect( (QObject *)programViewer, SIGNAL( pressed( int, int, int, const QPoint & )),
        this, SLOT( programViewerPressed( int, int, int, const QPoint & )));

    QObject::connect( (QObject *)programViewer, SIGNAL( selectionChanged() ),
        this, SLOT( programViewerSelectionChanged() ));

    QObject::connect( (QObject *)programViewer, SIGNAL( valueChanged( int, int )),
        this, SLOT( programViewerValueChanged( int, int )));

    // signal connections for address/data registers
    QObject::connect( (QObject *)registers, SIGNAL( currentChanged( int, int )),
        this, SLOT( registersCurrentChanged( int, int )) );

    QObject::connect( (QObject *)registers, SIGNAL( clicked( int, int, int, const QPoint & )),
        this, SLOT( registersClicked( int, int, int, const QPoint & )) );

    QObject::connect( (QObject *)registers, SIGNAL( doubleClicked( int, int, int, const QPoint & )),
        this, SLOT( registersDoubleClicked( int, int, int, const QPoint & )) );
```

```
QObject::connect( (QObject *)registers, SIGNAL( pressed( int, int, int, const QPoint & )),
    this, SLOT( registersPressed( int, int, int, const QPoint & )) );

QObject::connect( (QObject *)registers, SIGNAL( selectionChanged() ),
    this, SLOT( registersSelectionChanged() ));

QObject::connect( (QObject *)registers, SIGNAL( valueChanged( int, int )),
    this, SLOT( registersValueChanged( int, int )) );

// signal connections for other registers
QObject::connect( (QObject *)otherRegisters, SIGNAL( currentChanged( int, int )),
    this, SLOT( otherRegistersCurrentChanged( int, int )));

QObject::connect( (QObject *)otherRegisters, SIGNAL( clicked( int, int, int, const QPoint & )),
    this, SLOT( otherRegistersClicked( int, int, int, const QPoint & )) );

QObject::connect( (QObject *)otherRegisters, SIGNAL( doubleClicked( int, int, int, const QPoint & )),
    this, SLOT( otherRegistersDoubleClicked( int, int, int, const QPoint & )) );

QObject::connect( (QObject *)otherRegisters, SIGNAL( pressed( int, int, int, const QPoint & )),
    this, SLOT( otherRegistersPressed( int, int, int, const QPoint & )) );

QObject::connect( (QObject *)otherRegisters, SIGNAL( selectionChanged() ),
    this, SLOT( otherRegistersSelectionChanged() ));


QObject::connect( (QObject *)otherRegisters, SIGNAL( valueChanged( int, int )),
    this, SLOT( otherRegistersValueChanged( int, int )));

// signal connections for help stack
QObject::connect( (QObject *)helpStackTable, SIGNAL( currentChanged( int, int )),
    this, SLOT( helpStackTableCurrentChanged( int, int )));

QObject::connect( (QObject *)helpStackTable, SIGNAL( clicked( int, int, int, const QPoint & )),
    this, SLOT( helpStackTableClicked( int, int, int, const QPoint & )) );

QObject::connect( (QObject *)helpStackTable, SIGNAL( doubleClicked( int, int, int, const QPoint & )),
    this, SLOT( helpStackTableDoubleClicked( int, int, int, const QPoint & )));

QObject::connect( (QObject *)helpStackTable, SIGNAL( pressed( int, int, int, const QPoint & )),
    this, SLOT( helpStackTablePressed( int, int, int, const QPoint & )));

QObject::connect( (QObject *)helpStackTable, SIGNAL( selectionChanged() ),
    this, SLOT( helpStackTableSelectionChanged() ));

QObject::connect( (QObject *)helpStackTable, SIGNAL( valueChanged( int, int )),
    this, SLOT( helpStackTableValueChanged( int, int )));

// signal connections for stack
QObject::connect( (QObject *)stackTable, SIGNAL( currentChanged( int, int )),
    this, SLOT( stackTableCurrentChanged( int, int )));
```

```
QObject::connect( (QObject *)stackTable, SIGNAL( clicked( int, int, int, const QPoint & )),
    this, SLOT( stackTableClicked( int, int, int, const QPoint & )) );

QObject::connect( (QObject *)stackTable, SIGNAL( doubleClicked( int, int, int, const QPoint & )),
    this, SLOT( stackTableDoubleClicked( int, int, int, const QPoint & )));

QObject::connect( (QObject *)stackTable, SIGNAL( pressed( int, int, int, const QPoint & )),
    this, SLOT( stackTablePressed( int, int, int, const QPoint & )) );

QObject::connect( (QObject *)stackTable, SIGNAL( selectionChanged() ),
    this, SLOT( stackTableSelectionChanged() ));

QObject::connect( (QObject *)stackTable, SIGNAL( valueChanged( int, int )),
    this, SLOT( stackTableValueChanged( int, int )));

// signal connections for buttons
QObject::connect( (QObject *)open, SIGNAL( clicked() ),
    this, SLOT( openClicked() ));

QObject::connect( (QObject *)restart, SIGNAL( clicked() ),
    this, SLOT( restartClicked() ) );

QObject::connect( (QObject *)run, SIGNAL( clicked() ),
    this, SLOT( runClicked() ));

QObject::connect( (QObject *)undo, SIGNAL( clicked() ),
    this, SLOT( undoClicked() ));

QObject::connect( (QObject *)step, SIGNAL( clicked() ),
    this, SLOT( stepClicked() ));

QObject::connect( (QObject *)go, SIGNAL( clicked() ),
    this, SLOT( goClicked() ));

// Signal connections for the flags
QObject::connect( (QObject *)ledX, SIGNAL( clicked() ),
    this, SLOT( ledXClicked() ));

QObject::connect( (QObject *)ledN, SIGNAL( clicked() ),
    this, SLOT( ledNClicked() ));

QObject::connect( (QObject *)ledZ, SIGNAL( clicked() ),
    this, SLOT( ledZClicked() ));

QObject::connect( (QObject *)ledV, SIGNAL( clicked() ),
    this, SLOT( ledVClicked() ));

QObject::connect( (QObject *)ledC, SIGNAL( clicked() ),
    this, SLOT( ledCClicked() ));

// signal connections for spin box
QObject::connect( (QObject *)adjustSpeed, SIGNAL( valueChanged( int )),
```

```
        this, SLOT( adjustSpeedValueChanged( int )));

    // signal connections for options
    QObject::connect( (QObject *)decHex, SIGNAL( clicked() ),
        this, SLOT( decHexClicked() ));

    QObject::connect( (QObject *)decHex, SIGNAL( stateChanged( int )),
        this, SLOT( decHexStateChanged( int )));

    QObject::connect( (QObject *)autoScroll, SIGNAL( clicked() ),
        this, SLOT( autoScrollClicked() ));

    QObject::connect( (QObject *)autoScroll, SIGNAL( stateChanged( int )),
        this, SLOT( autoScrollStateChanged( int )));

    // Connect updateTables to the individual update slots
    QObject::connect( this, SIGNAL( updateTables() ), this, SLOT( memoryViewerUpdate() ));
    QObject::connect( this, SIGNAL( updateTables() ), this, SLOT( helpStackTableUpdate() ));
    QObject::connect( this, SIGNAL( updateTables() ), this, SLOT( programViewerUpdate() ));
    QObject::connect( this, SIGNAL( updateTables() ), this, SLOT( registersUpdate() ));
    QObject::connect( this, SIGNAL( updateTables() ), this, SLOT( stackTableUpdate() ));

    // Connect the run timer
    QObject::connect( (QObject *)timer, SIGNAL( timeout() ), this, SLOT( stepClicked() ));

    // Invoke the individual initialize methods
     helpStackTableInitialise ();
    memoryViewerInitialise();
    programViewerInitialise();
      registersInitialise ();
     stackTableInitialise ();

    // Set up What's This functionality
    DialogWhatsThis = new QWhatsThis( this );
     whatsThisInitialise ();

    // setup opcode bitmap
    opcodeBitmap = new QPixmap;
    setOpcodeBitmap("opcode");
    opcodeDefinition->setPixmap( *opcodeBitmap );
    opcodeDefinition->setAlignment( AlignLeft | AlignTop );
    opcodeDefinition->setProperty( "autoResize", "true" );

    CtrlPressed = false;
    pvCellEdited = false;
    m68008->decImm = !decHex->isChecked();

    emit updateTables();
}

void gui:: whatsThisInitialise ()
{
```

```
// Add the What's This text for any widget using the add method of the DialogWhatsThis object
// The What's This class accepts rich text, and the default 'style sheet' allows HTML tags
// to be used for formatting -- cool or what! :D See QStyleSheet...
// Note: normal escape characters doen't work as expected for formatted text, so for new line,
// use <br> instead of \n
DialogWhatsThis->add( (QWidget*)memoryViewer, "<b>Memory Viewer</b><br>Shows the contents of
    system memory.<br>Double-click an entry in the address column and type in the memory location
    you want the listing to start from.<br>Double-click on the contents of a memory location to change
    it's value." );
DialogWhatsThis->add( (QWidget*)helpStackTable, "<b>Help Stack</b><br>The last error or warning
    your code generated is shown here. Scroll down to see previous errors." );
DialogWhatsThis->add( (QWidget*)stackTable, "<b>System Stack Viewer</b><br>The address of items
    pushed to the system stack are shown here, along with the origin register or memory location of the
    item and the contents of the item.");
DialogWhatsThis->add( (QWidget*)programViewer, "<b>Program Viewer</b><br>Shows the location
    of instructions in memory, the instruction's opcode hex value and the interpretation of the instruction
    at the given memory location. Double click an entry in the address column to change the start
    address of the decoding listing." );
DialogWhatsThis->add( (QWidget*)registers, "<b>Address and Data Registers</b><br>The system's
    address and data registers are shown here.<br>Double-click on a cell to change it's value." );
DialogWhatsThis->add( (QWidget*)adjustSpeed, "<b>Run Speed</b><br>Change this value to alter
    the speed of program execution when Run is clicked. A large value gives a slow step speed and a
    small value gives a fast step speed.");
DialogWhatsThis->add( (QWidget*)otherRegisters, "<b>Other Registers</b><br>The PC shows the
    address of the instruction currently being executed.<br>The instruction register (IR) shows the
    opcode of the current instruction.<br>The CCR shows the status bits of the system.<br>Double-
    click a value to change it." );

// Buttons
DialogWhatsThis->add( (QWidget*)open, "<b>Open</b><br>Open a pre-assembled file." );
DialogWhatsThis->add( (QWidget*)restart, "<b>Reset</b><br>Reset the CPU." );
DialogWhatsThis->add( (QWidget*)run, "<b>Run</b><br>Step through the program automatically." );
DialogWhatsThis->add( (QWidget*)undo, "<b>Undo</b><br>Undo the effect of the last instruction." );
DialogWhatsThis->add( (QWidget*)step, "<b>Step</b><br>Execute the program instruction-by-
    instruction." );

// CCR LEDs
DialogWhatsThis->add( (QWidget*)ledC, "<b>Carry</b><br>Set if the result of an arithmetic
    operation causes a carry" );
DialogWhatsThis->add( (QWidget*)ledN, "<b>Negative</b><br>Set if the MSB of the result of an
    arithmetic operation is set (i.e. if the result is negative)" );
DialogWhatsThis->add( (QWidget*)ledV, "<b>Overflow</b><br>Set when the result of an arithmetic
    operation is outside the maximum number of bits the system can address, when interpreting the
    number as a signed number." );
DialogWhatsThis->add( (QWidget*)ledX, "<b>Extend</b><br>Used to extend the range of arithmetic
    operations to 64-bits" );
DialogWhatsThis->add( (QWidget*)ledZ, "<b>Zero</b><br>Set if the result of an instruction is zero."
    );

// Opcode Bitmap
DialogWhatsThis->add( (QWidget*)opcodeDefinition, "<b>Instruction Format</b><br>Shows the
    contents of the current instruction bit-by-bit." );
```

```
}

void gui::setOpcodeBitmap( const char* instrName )
{
    QString str( "./images/" );
    str.append( instrName );
    str.append( ".png" );
    opcodeBitmap->load( str );

    // using this to refresh the contents of the label -- is there a better way???
    opcodeDefinition->setPixmap( *opcodeBitmap );
}

void gui::keyPressEvent( QKeyEvent* e )
{
    if( e->key() == 4129 )
        CtrlPressed = true;
    e->ignore();
}

void gui::keyReleaseEvent( QKeyEvent* e)
{
    if( e->key() == 4129 )
        CtrlPressed = false;
    e->ignore();
}
```

1.21 qmtable.cpp

We have inherited and overloaded the QTable class to be able to implement highlighting. The definition for this class is in gui.h.

```
/*
    Motorola 68008 Simulator
    (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
    Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

    We derive a specialized QTable class which supports amongst other things
    row and cell highlighting .
*/

#include "guiBase.h"

#include <qpainter.h>
#include <qtable.h>
#include <qmessagebox.h>

void QMTable::paintCell( QPainter *p, int row, int col, const QRect & cr, bool selected )
{
// if ( selected && row == currentRow() && col == currentColumn() )
        selected = FALSE;
```

```
    int w = cr.width();
    int h = cr.height();
    int x2 = w - 1;
    int y2 = h - 1;

    QBrush brush;
    brush.setStyle( QBrush::SolidPattern );
    if ( row & 1 )
        brush.setColor( QColor( 247, 247, 255 ) );
    else
        brush.setColor( QColor( 237, 237, 255 ) );
    if ( ( ( row == hlRow )&& ( hlRows )) || ( ( ! hlRows ) && ( row == hlRow )&& ( col == hlCol )))
        brush.setColor( QColor( 255, 252, 84 ) );
    if ( ( ( row == hlRow2 )&& ( hlRows2 )) || ( ( ! hlRows2 ) && ( row == hlRow2 )&& ( col == hlCol2 )))
        brush.setColor( QColor( 255, 84, 84 ) );

    QTableItem *itm = item( row, col );
    if ( itm )
    {
        p->save();
        //itm->paint( p, colorGroup(), cr, selected );
        p->fillRect( 0, 0, w, h, brush );
        p->drawText( 2, 0, x2 - 4, y2, Qt::AlignVCenter | align[ col ], itm->text() );
        p->restore();
    }
    else
    {
        p->fillRect( 0, 0, w, h, selected ? colorGroup().brush( QColorGroup::Highlight )
            : colorGroup().brush( QColorGroup::Base ));
     }

    // Draw our lines
    QPen pen( p->pen() );
    p->setPen( colorGroup().mid() );
    p->drawLine( x2, 0, x2, y2 );
    p->drawLine( 0, y2, x2, y2 );
    p->setPen( pen );
}

void QMTable::keyPressEvent( QKeyEvent* e )
{
    if( e->key() == 4129 )
        CtrlPressed = true;
    e->ignore();
}

void QMTable::keyReleaseEvent( QKeyEvent* e )
{
    if( e->key() == 4129 )
        CtrlPressed = false;
    e->ignore();
```

```
}

void QMTable::setColumnsStretchable( int start, bool stretch )
{
    for ( int lp1 = start; lp1 < allCols; lp1++ )
        setColumnStretchable ( lp1, stretch );
}
```

1.22 buttons.cpp

Event handlers for the buttons (Go, Open, etc.). This includes the spinbox for the execution speed.

```
/*
    Motorola 68008 Simulator
    (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
    Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

    Graphical User Interface − Buttons
*/

#include "gui.h"
#include "../M68008.h"
#include "qapplication.h"
#include <qlabel.h>
#include <qfiledialog.h>
#include <qmessagebox.h>
#include <qfile.h>
#include <qspinbox.h>
#include <qpushbutton.h>

void gui::openClicked()
{
    unsigned long int initPC;

    QString s( QFileDialog::getOpenFileName(
        QString::null, "Compiled programs (*.h68)", this ));

        if ( s.isEmpty() )
    {
        QMessageBox::warning( this, "Motorola 68008 Simulator",
            "No .h68 file opened" );
        return;
    }

    FILE *in;
    in = fopen( s, "rw" );
    if ( !in )
    {
        QMessageBox::warning( this, "Motorola 68008 Simulator",
            "Could not open the file you selected" );
        return;
    }
```

```
    memory->LoadSRecords( in, &initPC );
    fclose ( in );

    m68008->SetInitPC( initPC );
    progBase = initPC;
    m68008->Reset();

    setProperty( "caption", s.prepend( "Motorola 68008 Simulator - " ));

    emit updateTables();
}

void gui::restartClicked()
{
    int sel;

    sel = QMessageBox::information( this, "Motorola 68008 Simulator",
        "Are you sure you want to reset?", "Yes", "No" );

    if( sel == 0 )
    {
        m68008->Reset();
        while ( systemStack.getCount() )
        {
            delete systemStack.pop();
        }
        emit updateTables();
    }
}

void gui::stepClicked()
{
    bool t = false;

    if( timer->isActive() )
    {
        t = true;
        timer->stop();
    }

    m68008->Step();
    if( m68008->IsHalted() )
    {
        QMessageBox::information( this, "Motorola 68008 Simulator",
            "CPU Halted" );
        run->setText( "Run" );
    }
    else if( breakPoint == m68008->pc )
    {
        QMessageBox::information( this, "Motorola 68008 Simulator",
            "Breakpoint hit" );
```

```
        run->setText( "Run" );
    }
    else
    {
        if( t )
            timer->start( adjustSpeed->property( "value" ).toInt(), false );
    }

    emit updateTables();
}

void gui::undoClicked()
{
    if ( !m68008->RestoreState() )
    {
        QMessageBox::warning( this, "Motorola 68008 Simulator", "Could not undo" );
        progBase = 0;
    }
    else
    {
        emit updateTables();
    }
}

void gui::runClicked()
{
    if( run->property( "text" ).toString() == "Run" )
    {
        run->setText( "Stop" );
        timer->start( adjustSpeed->property( "value" ).toInt(), false );
    }
    else
    {
        run->setText( "Run" );
        timer->stop();
    }
}

void gui::goClicked()
{
    int rv = 0;
    static bool dontAsk = false;

    if ( !dontAsk )
    {
        rv = QMessageBox::warning( this, "Motorola 68008 Simulator",
            "This will run the program <I>without</I> updating the GUI. This will run the program much
                faster, "
            "but you will not be able to see the effect of the program during the execution. "
            "Execution will stop if you press \"Stop\", when the CPU hits a breakpoint or when the CPU is
                halted. "
            "Undo will not be available for instructions executed in Go! mode. Do you want to continue?",
```

```
            "Yes", "No", "Yes, don't ask me again" );
    }

    if ( ( rv == 1 ) || ( rv == -1 ))
        return;

    if ( rv == 2 ) dontAsk = true;

    for ( ;; )
    {
        m68008->Step();
        if ( m68008->IsHalted() )
        {
            QMessageBox::information( this, "Motorola 68008 Simulator", "CPU is halted" );
            break;
        }
        if ( m68008->pc == breakPoint )
        {
            QMessageBox::information( this, "Motorola 68008 Simulator", "Breakpoint hit" );
            break;
        }
    }

    emit updateTables();
}

void gui::adjustSpeedValueChanged( int value )
{
    if ( timer->isActive() )
        timer->changeInterval( value );
}

void gui::ledXClicked()
{
    m68008->ccr.x ^= 1;
    emit updateTables();
}

void gui::ledNClicked()
{
    m68008->ccr.n ^= 1;
    emit updateTables();
}

void gui::ledZClicked()
{
    m68008->ccr.z ^= 1;
    emit updateTables();
}

void gui::ledVClicked()
{
```

```
    m68008->ccr.v ^= 1;
    emit updateTables();
}

void gui::ledCClicked()
{
    m68008->ccr.c ^= 1;
    emit updateTables();
}
```

1.23 programViewer.cpp

The program viewer.

```
/*
    Motorola 68008 Simulator
    (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
    Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

    Graphical User Interface - Program Viewer
*/

#include <stdlib.h>
#include <qtable.h>
#include <qmessagebox.h>
#include "gui.h"
#include "../M68008.h"

void gui::programViewerUpdate()
{
    static char lastInstr [ 20 ];
    static int lastRowHl = 0;

    unsigned long sizeOfFirstInstr = 0;
    char currentRow = -1;
    unsigned int lp1, lp2;
    unsigned long addr = progBase, newAddr;
    QString str, str2;
    char buf[ 32 ];

    // Reset highlighting
    programViewer->hlRow = -1;
    programViewer->hlRow2 = -1;

    for( lp1 = 0; lp1 < 128; lp1++ )
    {
        // Highlight this row if it's at the current point of execution
        if( addr == m68008->pc )
        {
            programViewer->hlRow = lp1;
            currentRow = lp1;
```

```
        if( autoScroll->isChecked() )
        {
            lastRowHl = lp1;

            if( lp1 > 0 )
                programViewer->ensureCellVisible( lp1 - 1, 0 );
            else
                programViewer->ensureCellVisible( lp1, 0 );

            programViewer->ensureCellVisible( lp1 + 2, 0 );
        }
    }

    // Highlight breakpoints
    if( addr == breakPoint )
    {
        programViewer->hlRow2 = lp1;
    }

    // Translate current instruction; get address of next one
    newAddr = m68008->Translate( addr, buf );

    // Check for decoding errors
    if( newAddr <= addr )
    {
        QMessageBox::warning( this, "Motorola 68008 Simulator",
            "Instruction Decoding Error" );
        break;
    }

    // Set size of first address
    if( lp1 == 0 )
        sizeOfFirstInstr = newAddr - addr;

    // Write the instruction address and translated string to the program viewer
    programViewer->setText( lp1, 0, str.sprintf( "$%08lX", addr ));
    programViewer->setText( lp1, 2, buf );

    // Write the opcode to the program viewer
    str2 = "$";
    for( lp2 = 0; lp2 < newAddr - addr; lp2++ )
    {
        str2.append( str.sprintf( "%02X", memory->GetByte( addr + lp2 )));
    }
    programViewer->setText( lp1, 1, str2 );

    // Move to the next instruction
    addr = newAddr;
}

// update the opcode bitmap only if it's different from the last instruction
if( strcmp( lastInstr, m68008->currentInstruction ))
```

```
{
    setOpcodeBitmap( m68008->currentInstruction );
    strcpy( lastInstr , m68008->currentInstruction );
}

// Make sure execution doesn't run off the end of the table -- 'scrolls' with 3 instructions to go
if ( ( currentRow < 0 ) && autoScroll->isChecked() && !pvCellEdited )
{
    // pc not in range of program viewer
    unsigned long addr = m68008->pc;

    // calc progBase so that the same row is highlighted, but it should now show
    // the current pc (which was previously out of range)
    for( int lp1 = 0; lp1 < lastRowHl; lp1++ )
    {
        newAddr = addr;

        while( m68008->Translate( newAddr, buf )>= addr )newAddr -= 2;
        addr = newAddr + 2;
    }

    progBase = addr;
    programViewerUpdate();
}
else if( autoScroll->isChecked() && !pvCellEdited )
{
    // stepping -- scrolling downwards...
    if( currentRow > programViewer->numRows() - 3 )
    {
        progBase += sizeOfFirstInstr;
        programViewerUpdate();
    }

    // undoing -- scrolling upwards...
    if( currentRow < 2 && m68008->undoStack.getCount() > 2 )
    {
        // reset the start row...
        progBase = m68008->undoStack.getItem( 0 )->pc;

        // do a quick update
        programViewerQuickUpdate( lastInstr );
    }
}

// Only skip an update for a direct edit once
if( pvCellEdited )
{
    // update the program viewer with the new value entered directly, but without
    // updating the program viewer relative to the currently highlighted row...
    for( int lp1 = 0; lp1 < lastRowHl; lp1++ )
    {
        newAddr = addr;
```

```
            while( m68008->Translate( newAddr, buf )>= addr )newAddr -= 2;
            addr = newAddr + 2;
        }

        progBase = addr;
        pvCellEdited = false;
    }
}

void gui::programViewerQuickUpdate( char* lastInstr )
{
    // Does some of the stuff as programViewerUpdate -- implemented for undoing; a call to
    // programViewerUpdate causes infinite recursion to take place :)

    // AD: lots of repeated code, I know -- will fix after the demo :)

    unsigned int lp1, lp2;
    unsigned long addr = progBase, newAddr;
    QString str, str2;
    char buf[ 32 ];

    for( lp1 = 0; lp1 < 128; lp1++ )
    {
        // Highlight this row if it's at the current point of execution
        if( addr == m68008->pc )
        {
            programViewer->hlRow = lp1;

            if( autoScroll->isChecked() )
            {
                if( lp1 > 0 )
                    programViewer->ensureCellVisible( lp1 - 1, 0 );
                else
                    programViewer->ensureCellVisible( lp1, 0 );

                programViewer->ensureCellVisible( lp1 + 2, 0 );
            }
        }

        // Highlight breakpoints
        if( addr == breakPoint )
        {
            programViewer->hlRow2 = lp1;
        }

        // Translate current instruction; get address of next one
        newAddr = m68008->Translate( addr, buf );

        // Check for decoding errors
        if( newAddr <= addr )
        {
```

```
            QMessageBox::warning( this, "Motorola 68008 Simulator",
                "Instruction Decoding Error" );
            break;
        }

        // Write the instruction address and translated string to the program viewer
        programViewer->setText( lp1, 0, str.sprintf( "$%08lX", addr ));
        programViewer->setText( lp1, 2, buf );

        // Write the opcode to the program viewer
        str2 = "$";
        for( lp2 = 0; lp2 < newAddr - addr; lp2++ )
        {
            str2.append( str.sprintf( "%02X", memory->GetByte( addr + lp2 )));
        }
        programViewer->setText( lp1, 1, str2 );

        // Move to the next instruction
        addr = newAddr;
    }

    // update the opcode bitmap only if it's different from the last instruction
    if( strcmp( lastInstr, m68008->currentInstruction ))
    {
        setOpcodeBitmap( m68008->currentInstruction );
        strcpy( lastInstr, m68008->currentInstruction );
    }
}
void gui::programViewerInitialise()
{
    progBase = 0;
    breakPoint = 0xFFFFFFFF;
    programViewer->hlRows = true;
    programViewer->hlRows2 = true;
}

void gui::programViewerCurrentChanged( int row, int col )
{
}

void gui::programViewerClicked( int row, int col, int button, const QPoint &mousePos )
{
    if( col > 0 && ( CtrlPressed || programViewer->CtrlPressed ))
    {
        QString str;
        unsigned long a;

        str = programViewer->text( row, 0 );
        str.replace ( 0, 1, " 0x" );
        a = strtoul( str, NULL, 0 );
        if( a == breakPoint )
            breakPoint = 0xFFFFFFFF;
```

```
        else
            breakPoint = a;
        emit updateTables();
    }
}

void gui::programViewerDoubleClicked( int row, int col, int button, const QPoint &mousePos )
{
    QString str;

    if( col > 0 )
    {
        str = programViewer->text( row, 0 );
        str.replace ( 0, 1, " 0x" );
        m68008->pc = strtoul( str, NULL, 0 );
        emit updateTables();
    }
}

void gui::programViewerPressed( int row, int col, int button, const QPoint &mousePos )
{
}

void gui::programViewerSelectionChanged()
{
    // Pass focus to the programViewer
    programViewer->setFocus();
}

void gui::programViewerValueChanged( int row, int col )
{
    QString str;
    unsigned long int addr, newAddr;
    char buf[ 32 ];

    // address
    if( col == 0 )
    {
        str = programViewer->text( row, col );
        if( str[ 0 ] == '$' )
            str.replace ( 0, 1, " 0x" );

        addr = strtoul( str, NULL, 0 );

        for( int lp1 = 0; lp1 < row; lp1++ )
        {
            newAddr = addr;

            while( m68008->Translate( newAddr, buf )>= addr )newAddr -= 2;
            addr = newAddr + 2;
        }
```

```
        progBase = addr;

        pvCellEdited = true;

        emit updateTables();
    }
}
// AD: Don't forget the extra line at the end to keep Solaris happy :-P
```

1.24 stackTable.cpp

The system stack viewer.

```
/*
    Motorola 68008 Simulator
    (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
    Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

    Graphical User Interface - Stack Viewer
*/

#include "gui.h"
#include "../M68008.h"
#include <qmessagebox.h>

void gui::stackTableUpdate()
{
    int lp1= 0;
    QString str;
    StackItem *current;
    unsigned long int data = 0;
    int id = 0, currentID;
    StackItem *array[8];
    int lp2 = 0;


    // initialises to blank
    for( lp1 = 0; lp1 < 128; lp1++ ){
        stackTable->setText( lp1, 0, "" );
        stackTable->setText( lp1, 1, "" );
        stackTable->setText( lp1, 2, "" );
    }

    if( systemStack.getCount() > 0 ){
        // This method will buffer the stack items with the same ID until it hits
        // one with a different ID when it will write it.
        for( lp1 = 0; lp1 < systemStack.getCount(); lp1++ ){
            // Get current

            current = systemStack.getItem( systemStack.getCount() - 1 - lp1 );
            currentID = current->ID;
```

```
    // if id != last
    //  store result into stackTable and move onto the next one while updating the array..
    if( id != currentID )
    {

        // It takes the data in the array and constructs the item to be displayed on the stack.
        stackTable->setText( id, 1, array[ 0 ]->origin );
        stackTable->setText( id, 0, str.sprintf( "$%08lX", array[ lp2 - 1 ]->address ));

        data = 0;
        for( int i = 0; i < lp2; i++ ){
            data = data | ( memory->GetByte( array[ i ]->address )<< (i * 8) );
        }

        // display a byte
        if( lp2 == 1 ){
            stackTable->setText( id, 2, str.sprintf( "$%02lX", data ));
        }

        // display a word
        if( lp2 == 2 ){
            stackTable->setText( id, 2, str.sprintf( "$%04lX", data ));
        }

        // display a 3-byte value
        if( lp2 == 3 ){
            stackTable->setText( id, 2, str.sprintf( "$%06lX", data ));
        }

        // display a long
        if( lp2 == 4 ){
            stackTable->setText( id, 2, str.sprintf( "$%08lX", data ));
        }


        // Resets the mechanism/
        lp2 = 0;
        id++;
    }

    array[ lp2++ ] = current;
}

// It takes the data in the array and constructs the item to be displayed on the stack.
stackTable->setText( id, 1, array[ 0 ]->origin );
stackTable->setText( id, 0, str.sprintf( "$%08lX", array[ lp2 - 1 ]->address ));

// gets the information from the array. Putting the pieces together again.
data = 0;
for( int i = 0; i < lp2; i++ ){
    data = data | ( memory->GetByte( array[ i ]->address )<< (i * 8) );
}
```

```
        // display a byte
        if( lp2 == 1 ){
            stackTable->setText( id, 2, str.sprintf( "$%02lX", data ));
        }

        // display a word
        if( lp2 == 2 ){
            stackTable->setText( id, 2, str.sprintf( "$%04lX", data ));
        }

        // display a 3-byte value
        if( lp2 == 3 ){
            stackTable->setText( id, 2, str.sprintf( "$%06lX", data ));
        }

        // display a long
        if( lp2 == 4 ){
            stackTable->setText( id, 2, str.sprintf( "$%08lX", data ));
        }
    }

/*
        stackTable->setText( id, 1, array[ 0 ]->origin );
        stackTable->setText( id, 0, str.sprintf( "$%08X", array[ lp2 - 1 ]->address ));

        data = 0;
        for( int i = 0; i < lp2; i++ ){
            data = data | ( memory->GetByte( array[ i ]->address )<< (i * 8) );
        }
        stackTable->setText( id, 2, str.sprintf( "$%08X", data ));
    }
*/

/*
    //for( lp1 = systemStack.getCount() - 1; lp1 > -1; lp1-- )
    for( lp1 = 0; lp1 < systemStack.getCount(); lp1++ )
    {
        current = systemStack.getItem( systemStack.getCount() - 1 - lp1 );

        stackTable->setText( lp1, 0, str.sprintf( "$%08lX", current->address ));
        stackTable->setText( lp1, 1, current->origin );

        switch( current->size ){
        case 1:
            stackTable->setText( lp1, 2, str.sprintf( "$%02X",
                memory->GetByte( current->address )));
            break;
        case 3:
            stackTable->setText( lp1, 2, str.sprintf( "$%04X",
                memory->GetWord( current->address )));
            break;
```

```
        case 2:
            stackTable->setText( lp1, 2, str.sprintf( "$%08lX",
                memory->GetLongword( current->address )));
            break;
        default:
            stackTable->setText( lp1, 2, "" );
            break;
        }


        stackTable->ensureCellVisible( lp1, 0 );
    }
*/
}

void gui:: stackTableInitialise ()
{
}

void gui::stackTableCurrentChanged( int row, int col )
{
}

void gui::stackTableClicked( int row, int col, int button, const QPoint &mousePos )
{
}

void gui::stackTableDoubleClicked( int row, int col, int button, const QPoint &mousePos )
{
}

void gui::stackTablePressed( int row, int col, int button, const QPoint &mousePos )
{
}

void gui::stackTableSelectionChanged()
{
    // Pass focus to the stackTable
    stackTable->setFocus();
}

void gui::stackTableValueChanged( int row, int col )
{
}
```

1.25 helpStack.cpp

The help stack viewer (the actual help stack messages are generated by the backend).

```
/*
    Motorola 68008 Simulator
    (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
```

```
   Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

   Graphical User Interface − Help Stack
*/

#include "gui.h"
#include <qtable.h>
#include <qmessagebox.h>
#include "../M68008.h"

void gui::helpStackTableUpdate()
{
   int lp1;
   static int oldCount = 0;
   QString str;

   for( lp1 = 0; lp1 < 128; lp1++ )
   {
      if( lp1 < helpStack.getCount() )
         helpStackTable->setText( lp1, 0, helpStack.getItem( lp1 ) );
      else
         helpStackTable->setText( lp1, 0, "" );
   }

   if( oldCount < helpStack.getCount() )
   {
      oldCount = helpStack.getCount();
      helpStackTable->hlRow = 0;
      QTimer::singleShot( 1000, this, SLOT( helpStackTableClearHl() ));
   }
}

void gui::helpStackTableClearHl()
{
   helpStackTable->hlRow = -1;
   helpStackTable->updateCell( 0, 0 );
}

void gui:: helpStackTableInitialise ()
{
   helpStackTable->hlRows = true;
}

void gui::helpStackTableCurrentChanged( int row, int col )
{
}

void gui::helpStackTableClicked( int row, int col , int button, const QPoint &mousePos )
{
/*
   // make sure focus is passed to widget when left-clicked
   if( button == 1 && !helpStackTable->hasFocus() )
```

```
    helpStackTable->setFocus();
*/
}

void gui::helpStackTableDoubleClicked( int row, int col, int button, const QPoint &mousePos )
{
}

void gui::helpStackTablePressed( int row, int col, int button, const QPoint &mousePos )
{
}

void gui::helpStackTableSelectionChanged()
{
}

void gui::helpStackTableValueChanged( int row, int col )
{
}
```

1.26 memoryViewer.cpp

The memory viewer.

```
/*
   Motorola 68008 Simulator
   (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
   Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

   Graphical User Interface - Memory Viewer
*/

#include "gui.h"
#include "../M68008.h"
#include <qtable.h>
#include <qmessagebox.h>
#include <stdlib.h>

void gui::memoryViewerUpdate()
{
   int lp1, lp2;
   unsigned long addr = memBase;
   QString str;

   for( lp1 = 0; lp1 < 128; lp1++ )
   {
      memoryViewer->setText( lp1, 0, str.sprintf( "$%08lX", addr ));
      for( lp2 = 0; lp2 < 16; lp2++ )
      {
         memoryViewer->setText( lp1, 1 + lp2,
            str.sprintf( "$%02X", memory->GetByte( addr )));
         addr++;
```

```
        }
    }
}

void gui::memoryViewerInitialise()
{
    memBase = 0;
}

void gui::memoryViewerCurrentChanged( int row, int col )
{
}

void gui::memoryViewerClicked( int row, int col, int button, const QPoint &mousePos )
{
}

void gui::memoryViewerDoubleClicked( int row, int col, int button, const QPoint &mousePos )
{
}

void gui::memoryViewerPressed( int row, int col, int button, const QPoint &mousePos )
{
}

void gui::memoryViewerSelectionChanged()
{
    // Pass focus to the memoryViewer
    memoryViewer->setFocus();
}

void gui::memoryViewerValueChanged( int row, int col )
{
    QString str;

    // address
    if( col == 0 )
    {
        str = memoryViewer->text( row, col );
        if( str[ 0 ] == '$' )
            str.replace ( 0, 1, " 0x" );
        memBase = strtoul( str, NULL, 0 )- row * 16;
        emit updateTables();
    }

    // data
    if( col != 0 )
    {
        str = memoryViewer->text( row, col );
        if( str[ 0 ] == '$' )
            str.replace ( 0, 1, " 0x" );
        //QMessageBox::information( this, NULL, str );
```

```
        memory->SetByte( strtoul( str, NULL, 0 ), memBase + row * 16 + col - 1 );
        emit updateTables();
    }
}
```

1.27 register.cpp

The registers; data registers, address registers, PC, IR and flags including LEDs.

```
/*
    Motorola 68008 Simulator
    (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
    Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

    Graphical User Interface - Register Viewer
*/

#include <stdio.h>
#include <stdlib.h>
#include <qlabel.h>
#include <qpushbutton.h>
#include <qtable.h>
#include <qmessagebox.h>
#include "gui.h"
#include "../M68008.h"
// handles both tables

void gui::registersUpdate()
{
    int ccr, lp1;
    QString str;

    ccr = m68008->ccr.c + ( m68008->ccr.v << 1 )+ ( m68008->ccr.z << 2 )+
        ( m68008->ccr.n << 3 )+ ( m68008->ccr.x << 4 );

    for( lp1 = 0; lp1 < 8; lp1++ )
    {
        registers->setText( lp1, 0, str.sprintf( "$%08lX", m68008->d[ lp1 ].l ));
        registers->setText( lp1, 1, str.sprintf( "$%08lX", m68008->a[ lp1 ].l ));
    }

    otherRegisters->setText( 0, 0, str.sprintf( "$%08lX", m68008->pc ));
    otherRegisters->setText( 1, 0, str.sprintf( "$%04X", m68008->ir ));
    otherRegisters->setText( 2, 0, str.sprintf( "%c%d%d%d%d%d ($%02X)", '%',
        m68008->ccr.x, m68008->ccr.n, m68008->ccr.z, m68008->ccr.v, m68008->ccr.c,
        ccr ) );

    // Update the flags
    if( m68008->ccr.x )
        ledX->setPalette( QPalette( QColor( 40, 255, 40 )) );
    else
        ledX->setPalette( QPalette( QColor( 255, 40, 40 )) );
```

```
    if ( m68008->ccr.n )
        ledN->setPalette( QPalette( QColor( 40, 255, 40 )) );
    else
        ledN->setPalette( QPalette( QColor( 255, 40, 40 )) );

    if ( m68008->ccr.z )
        ledZ->setPalette( QPalette( QColor( 40, 255, 40 )) );
    else
        ledZ->setPalette( QPalette( QColor( 255, 40, 40 )) );

    if ( m68008->ccr.v )
        ledV->setPalette( QPalette( QColor( 40, 255, 40 )) );
    else
        ledV->setPalette( QPalette( QColor( 255, 40, 40 )) );

    if ( m68008->ccr.c )
        ledC->setPalette( QPalette( QColor( 40, 255, 40 )) );
    else
        ledC->setPalette( QPalette( QColor( 255, 40, 40 )) );
}

void gui:: registersInitialise ()
{
}

// Data and address registers

void gui::registersCurrentChanged( int row, int col )
{
}

void gui:: registersClicked ( int row, int col, int button, const QPoint &mousePos )
{
}

void gui::registersDoubleClicked( int row, int col, int button, const QPoint &mousePos )
{
}

void gui:: registersPressed ( int row, int col, int button, const QPoint &mousePos )
{
}

void gui::registersSelectionChanged()
{
    // Pass focus to the registers
    registers->setFocus();
}

void gui::registersValueChanged( int row, int col )
{
```

```
    QString str;

    // data
    if( col == 0 )
    {
        str = registers->text( row, col );
        if( str[ 0 ] == '$' )
            str.replace ( 0, 1, " 0x" );
        m68008->d[ row ].l = strtoul( str, NULL, 0 );
        emit updateTables();
    }

    // address
    if( col == 1 )
    {
        str = registers->text( row, col );
        if( str[ 0 ] == '$' )
            str.replace ( 0, 1, " 0x" );
        m68008->a[ row ].l = strtoul( str, NULL, 0 );
        emit updateTables();
    }
}

// PC, IR and CCR

void gui::otherRegistersCurrentChanged( int row, int col )
{
}

void gui::otherRegistersClicked( int row, int col, int button, const QPoint &mousePos )
{
}

void gui::otherRegistersDoubleClicked( int row, int col, int button, const QPoint &mousePos )
{
}

void gui::otherRegistersPressed( int row, int col, int button, const QPoint &mousePos )
{
}

void gui::otherRegistersSelectionChanged()
{
    // pass keyboard focus to other registers
    otherRegisters->setFocus();
}

void gui::otherRegistersValueChanged( int row, int col )
{
    QString str;

    // PC
```

```
    if( col == 0 )
    {
        str = otherRegisters->text( 0, 0 );
        if( str[ 0 ] == '$' )
            str.replace ( 0, 1, " 0x" );
        m68008->pc = strtoul( str, NULL, 0 );
        progBase = m68008->pc;
        emit updateTables();
    }
}
```

1.28 options.cpp

The options for decimal/hexadecimal display and automatic scrolling for the program viewer.

```
/*
    Motorola 68008 Simulator
    (C) 2002 The 68k/Sim Team (Diarmuid Power, Annie Bedford, Laura Redmond,
    Alan Donnelly, Gerard Whyte and Edsko de Vries). All rights reserved.

    Graphical User Interface - Program Options
*/

#include <stdlib.h>
#include <qtable.h>
#include <qmessagebox.h>
#include "gui.h"
#include "../M68008.h"

void gui::decHexClicked()
{
}

void gui::decHexStateChanged( int state )
{
    m68008->decImm = !state;
    emit updateTables();
}

void gui::autoScrollClicked()
{
}

void gui::autoScrollStateChanged( int state )
{
    emit updateTables();
}
```