ELEC 464 : Microcomputer System Design
Fall 1995

Review of Motorola 68000 Architecture and Instruction Set

Although you need not be familiar with every 68000 instruction, you should be able to use the tables in the Appendix to look up the details of every instruction. You should also be sufficiently familiar with the 68000 architecture and the basic instructions (move, arithmetic/logic and conditional branch) to write simple routines.

The Motorola 68000 Internal Architecture

The Motorola 68000 has eight 32-bit data registers (D0 to D7) used for general arithmetic and logic operations and eight 32-bit address registers (A0 to A7) used for address computations. Each data register can be used for byte, word (16-bit) and long word (32 bit) operations. Address register A7 is also used as the stack pointer for call and return instructions. There are actually two stack pointers, one for normal user software (USP) and another (SSP) ("supervisor mode stack pointer") for use by the operating system and exception handlers.

The 68000 also has a program counter (PC) register and a status register (SR). The least significant byte of the SR contains the condition code register (CCR) with bits (flags) to indicate overflow, parity, etc which are set by logical and arithmetic instructions.

Endianness

The 68000 is a *big endian* processor. This means that word and longword values are stored in successive byte locations in order from most significant to least significant byte. For example, if the 16-bit hexadecimal value $ABCD were stored at address 0, address 0 would contain AB and address 1 would have CD. Other computers, such as the Intel 80x86 series are *little endian* and store values in order from least significant to most significant byte.

Exercise: If the the 4-byte value $01020304 were stored at memory location $1000 by a 68000, what would be the values of the bytes in locations $1000 to $1003? What if the value was stored by an Intel 386?

MC68000 Instruction Set

The 68000 instructions include the following broad instruction types:

- move - moves data between memory locations and/or registers (`MOVE D0,D3`)
- arithmetic - add, subtract, multiply, divide, clear, negate, compare (`ADD D0,D3`)
- logical - and, or, exclusive or, not, (`AND D0,D3`)
- shift/rotate - arithmetic or logical (`ASL 4,D0`)
- branch, jump - conditional branch or unconditional jump (`BEQ LOOP`) (`JMP RETRY`)
- subroutine call/return - branch (PC relative) or jump (`JSR PRINT`) (`RTS`)

The MC68000 Addressing Modes

The more important 68000 addressing modes (with examples in assembly language) include:

- Immediate - the operand is included in the instruction (`MOVE #1,D3`)
- Absolute - the address of the operand is given (`MOVE $100,D3`)
- Register - the operand is a register (`MOVE D0,D3`)

- Register Indirect - an address register gives the address of the operand. The register can also be decremented before use or incremented after use to implement stacks and access consecutive data values. (`MOVE (A5),D3`), (`MOVE -(A5),D3`), (`MOVE (A5)+,D3`)
- Register Indirect with offset - as above but with an additional constant offset contained in the instruction (`MOVE 4(A5),D3`)

The 68000 Programmer's Reference Card

The various tables in this appendix summarize the architecture and instruction set of the 68000.

The largest table (pp 875–877) is a summary of the 68000's instructions. It shows the following for each instruction:

- the opcode mnemonic (`ADD`)
- a table of allowed source and destination addressing modes
- the number of bytes and clock cycles required
- the encoding of the 16-bit machine-language opcode
- a symbolic description of the instruction's effect
- a table of which condition code bits are affected

For example, consider the instruction `MOVE.B $FFFF102C,D3`. The destination (d) is D3 so the addressing mode is data register direct ("s=Dn"). The source (s) is a 32-bit address so the address mode is absolute long ("Abs.L"). The appropriate row from the table on page 876 (the first row for MOVE instructions) shows that this instruction is 6 bytes long and takes 16 clock cycles to execute. The value of the opcode is given by 00XX RRR MMM eee eee. From the footnotes on page 877 and the tables on page 875, and 878, XX is 01, RRR is 011 (register 3), MMM is 000 (register direct) and eee eee (effective address mode and register) is 111 001.

The resulting bit pattern for the instruction is: 00 01 011 000 111 001 = 0001 0110 0011 1001 = $1639. This 16-bit instruction word would be followed by the 32-bit source address with the most-significant word first: $FFFF $102c.

The final column shows which CCR flags are affected. X (extend) is not affected, N and Z (negative and zero) are set according to the value stored while V and C (overflow and carry) are cleared.

Exercise: Obtain the above information for the instructions `SUB #1,D0` and `BEQ LOOP` (assume `LOOP` is 32 bytes *before* the next instruction).

Assembler Directives

The `ORG` pseudo-opcode is used to tell the assembler the location where the assembled code will be placed when it is run. The `DS` (define storage) directive reserves memory. The `DC` directive places constant data in memory. The `END` directive marks the end of the program and give the program start address.

Example

The following subroutine sets an area of memory to the value $FFFF.

```
        ORG     $1000
MEMSET: MOVE.W  #$FFFF,(A0)+
        SUBI    #1,D0
        BNE     MEMSET
        RTS
        END     MEMSET
```

Exercise: Identify the addressing modes used in each of the above instructions. How does this subroutine work? What registers need to be set up before calling this routine? Write a short section of code that could be used to set locations from $2000 to $2003 to $FF (Hint: use `JSR MEMSET` to call the function).