68000 Instruction Set

Lecture Overview

- The 68000 Instruction Set
- The understand and effectively use an architecture must understand the register set and the **instruction set.**

Instruction Grouping

- Instructions can be grouped into classes
  - Data movement
  - Arithmetic operations
  - Logical operations
  - Shift operations
  - Bit manipulations
  - Program control

Condition Code Register

- Execution of the various instructions will set different bits in the condition code register
- A table is provided to show which instructions affect which status bits
- For the 68000, Table 2.2 of the text gives this information (pages 47-49) – Legend is at the end of the table.

Data Movement

- General form: MOVE source,destination
- MOVE MOVE #$00123456, D7
  - Source operand – any addressing mode – Data in register, or memory location
  - Destination – use any addressing mode except immediate addressing, program counter relative
  - V & C bits of CCR are cleared, N & Z updated
- MOVEA – Move address
  - Destination must be an address register (does not affect CCR)

DATA Movement (2)

- MOVE to CCR
  - Move data into the CCR
  - MOVE <ea>, CCR
    - Copies the low order byte at <ea> to the low order byte of the status register.
    - High order byte (status register) unaffected

MOVE to SR

- MOVE to or from SR
  - TO the status register is a privledged instruction – only when in supervisor mode.
  - FROM can be done in either mode on the 68000 but only supervisor mode on the 68010, 68020, or 68030.
  - Form
    - MOVE <ea>,SR
    - MOVE SR,<ea>

MOVE USP

- There are actually two A7 registers, one for the user and one that is used in supervisor mode (USP and SSP)
- MOVE.L USP,An **or** MOVE.L An,USP copy USP value to/from address register An
- Register other than the USP must be another address register.
- User cannot see system stack pointer, SSP

MOVEM

- Move multiple registers – allows movement of multiple register to/from memory
  - MOVEM <register list>,<ea>
  - MOVEM <ea>,<register list>
- Example
  - MOVEM.L D0-D7/A0-A6, -(SP)
    - Pushes all data and all address except A7 onto stack
- When would this be useful?

MOVEQ

- Move quick
  - Load a value in the range -128 to +127 into a data register
  - Loaded as a 32 bit value
    - The value is sign extended to a 32-bit value
  - MOVEQ #-3, D2
    - Loads $FFFF FFFD into D2

MOVEP

- Move peripheral instruction
- Copies words or longwords to an 8 bit peripheral
- Byte orientated peripheral interfaced such that they map to odd or even byte addresses.
- MOVEP.L D2,0(A0)
  - Copies contents of D2 to [A0+0], [A0+2], [A0+4], [A0+6]

LEA

- Load Effective Address
  - Calculate and effective address and then load it into an address register
- LEA $0010FFFF,A6

- LEA $12(A0,D4.L), A5

Position independent destinations

- PC relative only allowed for source operands
- Consider
  - LEA TABLE(PC),A0
  - MOVE.B D0,(A0)
  - Loads the position independent address of TABLE into address register A0
  - Then the least significant byte of D0 is stored at the address indicated by A0

PEA

- Push effective address
- Calculates the effective address and pushes it to the stack
- Useful to put calling routines address on stack so subroutine can access actual data.

Example of PEA

- Main routine (calling routine)

```
P1      DS.W        1           Parameter P1
        …
        PEA         P1          Push addr of P1 on stack
        BSR         ABC         Jump to subroutine ABC
        LEA         4(A7),A7    Incr stack ptr back

ABC     LEA         4(A7),A0    Load addr on stack of
                                of P1 into A0
        MOVEA.L     (A0),A0     Put P1 addr into A0
        MOVE.W      (A0),D0     Get data at P1
```

EXG & SWAP

- EXG
  - Exchange the contents of two registers
  - Exchanges the entire 32-bit contents
- SWAP
  - Exchange the high and low order words of a data register.

Integer Arithmetic Operations

- Conventional set of integer arithmetic ops
- Act on 8, 16, or 32 bit operands

- ADD - add source and destination and place result in destination
  - Both can be data registers OR
  - At least one must be a data register

Arithmetic

- ADDA – destination of add is an address register
- ADDQ – add a literal value in the range 1 to 8 to the contents of a memory location or register.
  - ADDQ #4, D1
  - Speed is faster than ADD #4, D1

Arithmetic

- ADDI - Add immediate – adds a literal value of a byte, word, or longword to the contents of a destination operand and then stores the result in the destination.
  - ADDI.W #1234,(A0)
  - Cannot be done using ADD as one operand must be a data register. Here can do without a data register.
- ADDX – Add extended
  - Add source and destination plus contents of the X bit of the condition code register
  - Both source and destination must be data registers
  - Carry out of msb is stored in X from operations so this allows mutiprecision data.

ARITHMETIC

- CLR – Loads the target with 0

- DIVS, DIVU – Integer division, signed or unsigned
  - DIVU <ea>,Dn -32-bit longword in Dn is divided by the low order 16 bits at <ea>. Quotient is 16-bits and depostied in low-order word of destination

ARITHMETIC

- MULS, MULU – multiply signed or unsigned
- SUB, SUBA, SUBQ, SUBL, SUBX – the subtractions equivalents of ADD
- NEG – Take the 2's complement of target
- NEGX – Two's complement with X bit
- EXT – sign extend low-order byte for word, or word for longword of destination – valid for .W and .L
  - DIVX D0,D1 Divide 32 bit val in D1 by D0
  - EXT.L D1

Lecture Summary

- Have looked at data movement and arithmetic instructions of the 68000.