

# Programming in Assembly

Data movement,  
Arithmetic,  
Logical Instructions,  
Shift Operations

## MOVEQ

Sign-extend an 8-bit value to 32 bits and copy to a data register

```
MOVEQ    #-3,D2 ;[D2] B FFFFFFFD
```

Why is it “quicker” than MOVE.L?

## MOVEA

Copy a sign-extended 32-bit value to address register

```
MOVEA.W  #$8C00,A0 ;[A0] B FFFF8C00
```

Compare to

```
MOVEA.L  #$8C00,A0 ;[A0] B ?
```

# Data Movement

## MOVE

Copy an 8-, 16-, or 32-bit value from memory or register to memory or register

```
MOVE.B   #$80,D0
```

## MOVE to CCR

To move a data to the condition code register

```
MOVE    #%00001,CCR
```

## MOVE to/from SR (Status Register)

Copy data to/from status register.

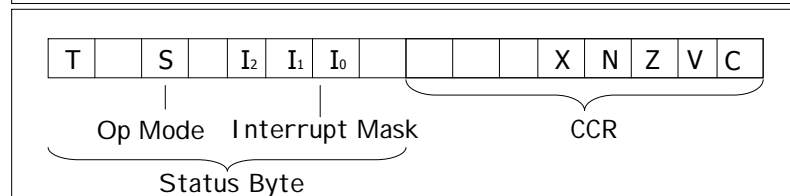
```
MOVE    SR,D3
```

Note: Move to SR is privileged instruction.

In SR, S=0 means operation under user mode;

S=1 means operation under supervisor mode

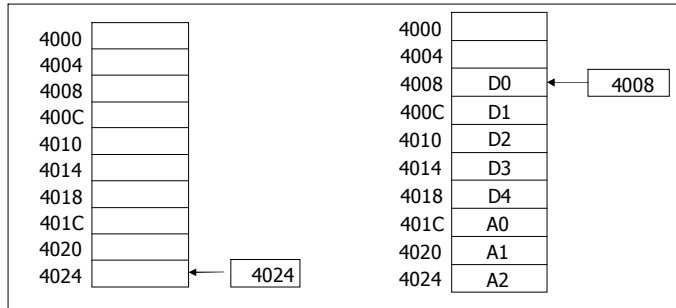
T: Trace, T = 1 means diagnostic Mode



## MOVEM

Transfer the contents of a group of registers to consecutive memory locations

```
MOVEM.L    D0-D4/A0-A2, -(A7)
...
MOVEM.L    (A7)+, D0-D4/A0-A2
```



68000 lecture notes

5

## EXG

Exchange the entire 32-bit contents of two registers

```
; [A4] B 10 * [A4] + 6
EXG      A4,D2      ;preserve D2
MULU    #10,D2     ;x10
ADD.L   #6,D2      ;+6
EXG      A4,D2      ;retreat D2
```

## SWAP

SWAP Dn

Exchanges the upper- and lower-order words of register Dn

If [D1]=\$1234567, after **SWAP D1**, [D1]=\$45670123

68000 lecture notes

7

## REG: define register list

```
ORG      $400400

SAVE_ALL: REG      A0-A6/D0-D7
SAVE_FEW: REG      A0-A1/A4-A5/D0-D2/D7
...
PROCA:   MOVEM.L   SAVE_ALL, -(A7)
...
        MOVEM.L   (A7)+, SAVE_ALL
        RTS
...
PROCB:   MOVEM.L   SAVE_FEW, -(A7)
...
        MOVEM.L   (A7)+, SAVE_FEW
        RTS

        END
```

68000 lecture notes

6

## LEA: Lead Effective Address

Copy an effective address into an address register.

```
Position Independent Code
        LEA      Table1(PC), A0
        LEA      Table2(PC), A1
        MOVE.B   #12, D0
Loop    MOVE.B   (A0)+, D1
        ADD.B   D1, (A1)+
        SUB.B   #1, D0
        BNE     Loop
...
Table1  DS.B    12
Table2  DS.B    12
```

68000 lecture notes

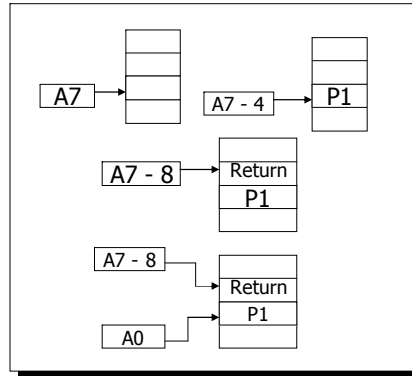
8

# PEA

Calculate an effective address and pushes it onto the stack.  
Parameter passing using the stack.

```

P1  DS.W  1
...
PEA   P1
BSR   ABC
LEA   (4,SP),SP
...
ABC  LEA   (4,SP),A0
      MOVEA.L (A0),A0
      MOVE.W  (A0),D0
      ...
      RTS
    
```



68000 lecture notes

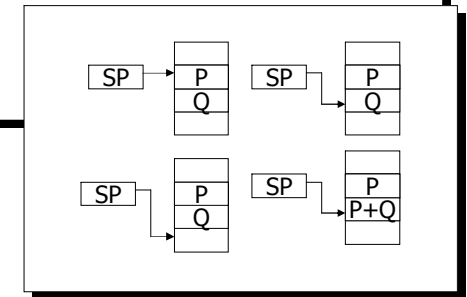
9

# ADD

**NO direct memory-to-memory additions allowed.**

```

MOVE.W  (SP)+,D0
ADD.W   (SP)+,D0
MOVE.W  D0,-(SP)
    
```



68000 lecture notes

11

# Arithmetic Operations

Add, subtract, sign extension, etc

68000 lecture notes

10

# ADDA

```
ADDA.W  #$8122,A0
```

Sign-extend \$8122 to \$FFFF8122 before add it to 32-bit A0. Compare to:

```
ADDA.L  #$8122,A0
```

# ADDQ

```
ADDQ.W  #4,D1
```

Add a constant (1 to 8) to a memory location, or register. It is executed faster. Why is it faster than :

```
ADD.W   #4,D1
```

68000 lecture notes

12

## ADDI

```
ADDI.W #1234, (A0)
```

Add an immediate value.

```
ADDI.W #1234,D4  
ADD.W #1234,D4
```

are equivalent But "ADDI #" can add to a memory location

## ADDX

Like ADD instruction, but also add X-bit. Used for extended arithmetic.

## CLR

Clear a data register or memory location.

## EXT: Sign Extension

- Sign extend. Often used with DIVS, because DIVS requires a 32-bit dividend.
- **EXT.L** D1 sign-extends the low-order word in D1 to 32 bits by copying D1(15) to bits D1(16:31).

```
MOVE.W (A0),D0
```

```
EXT.L D0
```

```
DIVS #42,D0
```

```
MOVE.W D0,2(A0)
```

- **EXT.W** D1 sign-extends the low-order byte in D1 to 16 bits by copying D1(7) to bits D1(8:15).

## Division: DIVU, DIVS

DIVU: unsigned division  
DIVS: signed division

Quotient (16 bits): in the lower 16 bits of a register  
Remainder: in the upper 16 bits of a register

If [D0]=\$00005678, after `DIVU #1234,D0` [D0]=\$0DA80004

How do we delete the remainder in D0?

```
AND.L #0000FFFF,D0
```

How do we obtain the remainder?  
Use SWAP

## Example: Greatest Common Divisor

```
GCD(A,B)  
IF A < B THEN  
  SWAP(A, B)  
WHILE ((R = (A MOD B)) != 0) DO  
  A = B  
  B = R  
GCD = B  
END
```

<pre> ORG \$400400 CLR.L D0 MOVE.W A,D0 MOVE.W B,D1 CMP.W D1,D0 BGE NEXT MOVE.W D0,D2 MOVE.W D1,D0 MOVE.W D2,D1 NEXT EXT.L D0 DIVU D1,D0 SWAP D0 TST.W D0 BEQ DONE MOVE.W D0,D2 MOVE.W D1,D0 MOVE.W D2,D1 BRA NEXT DONE MOVE.W D1,GCD ..... </pre>	<pre> ORG \$400800 A DS.W 1 B DS.W 1 GCD DS.W 1 END \$400 </pre>
--	--

## NEG

Calculate the 2's complement of an operand

```

MODULUS TST.L D0
          BPL Exit
          NEG.L D0
Exit     RTS

```

## NEGX

Calculate the 2's complement of an operand minus the X-bit. Used for extended arithmetic.

## MULS, MULU

- **MULS**: multiply two 16-bit integers represented in 2's complement.
- **MULU**: multiply two 16-bit unsigned integers.

If [D1]=\$ABCD5678

```
MULU.W #$1234,D1
```

Then [D1]=\$6260060 as only low-order word of D1 is used in multiplication

## SUB, SUBA, SUBQ, SUBI, SUBX

```
SUBI.B #$12,D1 [D0(0:7)] <- [D0(0:7)] - $12
```