Chapter 4

68000 Instruction Sets

Expected Outcomes

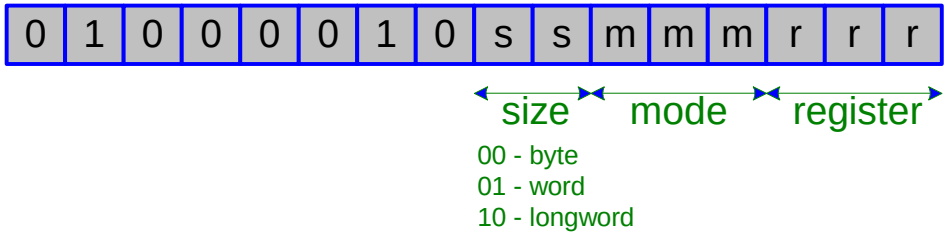
- Analyze and interpret the simple instruction set and addressing mode
- Use various form of instruction sets in a program
- Infer the outcome of flags in the status register

CLR Instruction

- Operation : Clear data in a location
- Format instruction set
  - **Assembly language**

```
CLR.s        destination
```

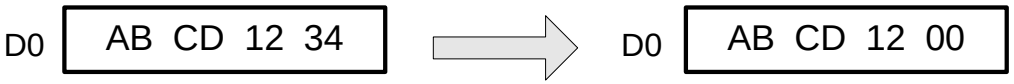
  - **Machine Code**



CLR Instruction

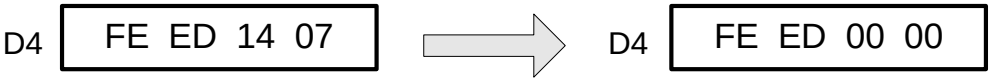
- **Example:**

Assembly Code	Machine Code
`CLR.B D0`	`$4200`



- **Example:**

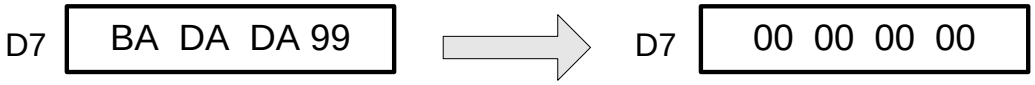
Assembly Code	Machine Code
`CLR.W D4`	`$4244`



CLR Instruction

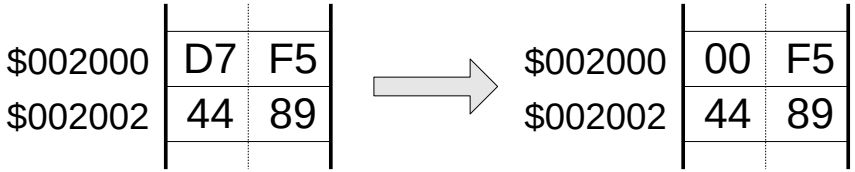
- **Example:**

Assembly Code	Machine Code
`CLR.L D7`	`$4287`



- **Example** (using Absolute Short mode)

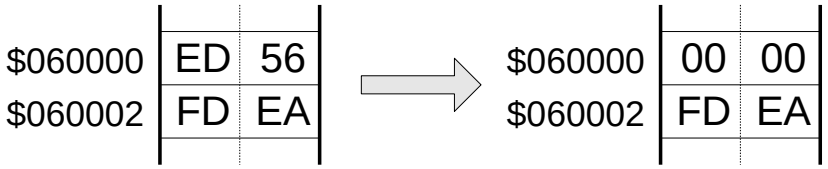
Assembly Code	Machine Code
`CLR.B $2000`	`$4238 2000`



CLR Instruction

- **Example** (Using Absolute Long mode)

Assembly Code	Machine Code
`CLR.W $60000`	`$4279 0006 0000`



CLR Instruction

- **Example** (Illegal operation)

```
CLR.W    $2001
```

Reason: *Word operation must begin at even address*

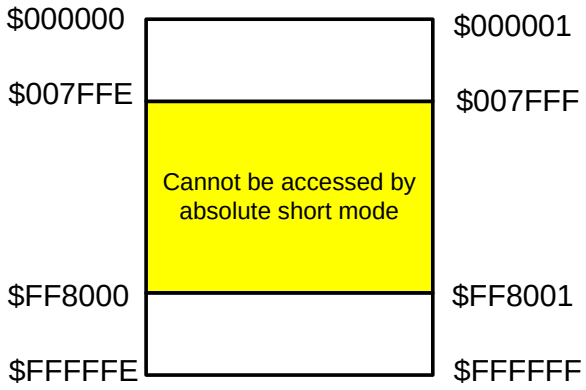
- **Example** (Illegal operation)

```
CLR.L    A1
```

Reason: *Special instruction is required to access address register*

Absolute Short & Long

- Address memory has 16 bits with sign extended mode is applied; that is the sign of 15th bit is extended to 16-23 bits during accessing the memory
- Thus, the range of memory address can be accessed using **absolute short** mode is between $000000 to $007FFF and $FF8000 to $FFFFFF
- If a user try to access the location of $9000, he must use the **absolute long** mode operation as this mode allows the user to access any locations

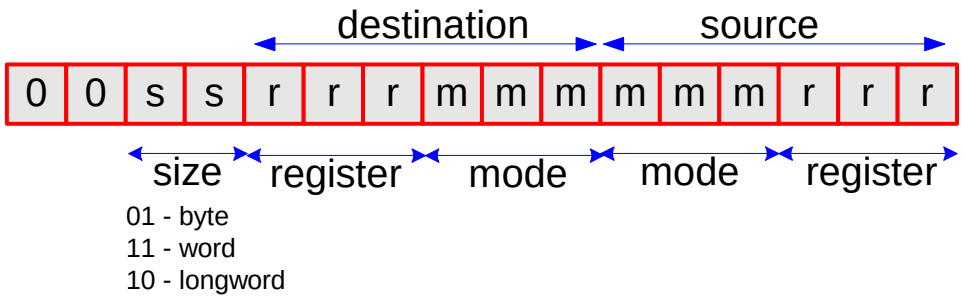


MOVE Instruction

- **Operation** : To copy or transfer data from source to destination
- **Format Instruction set**
  - **Assembly language**

```
MOVE.s      source,destination
```

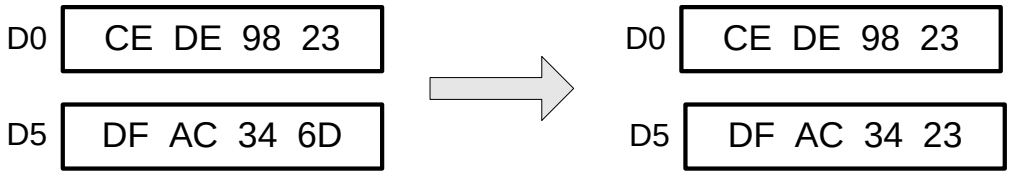
  - **Machine Code**



MOVE Instruction

- **Example:**

Assembly Code	Machine Code
`MOVE.B D0,D5`	`$1A00`



MOVE Instruction

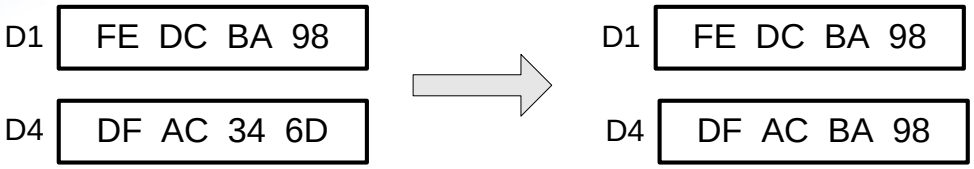
- **Example:**

Assembly Code

```
MOVE.W D1,D4
```

Machine Code

```
$3801
```



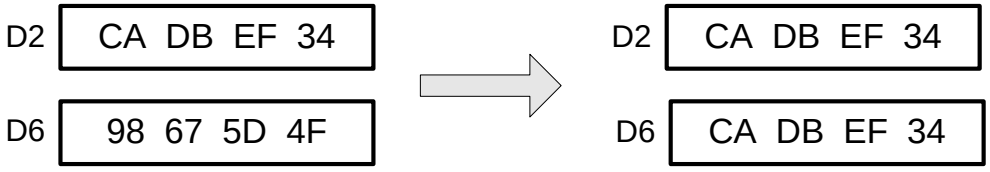
- **Example:**

Assembly Code

```
MOVE.L D2,D6
```

Machine Code

```
$2C02
```



MOVE Instruction

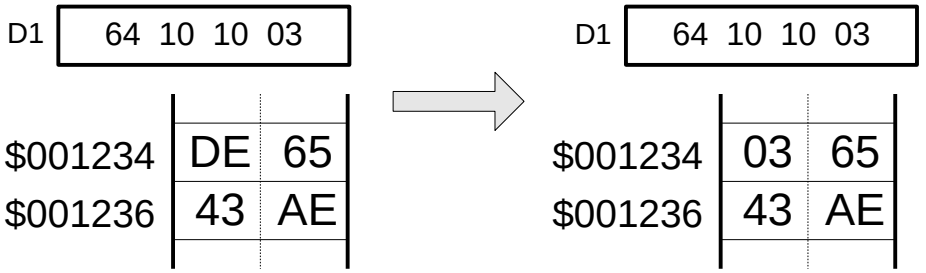
- **Example:**

Assembly Code

```
MOVE.B D1,$1234
```

Machine Code

```
$11C1 1234
```



MOVE Instruction

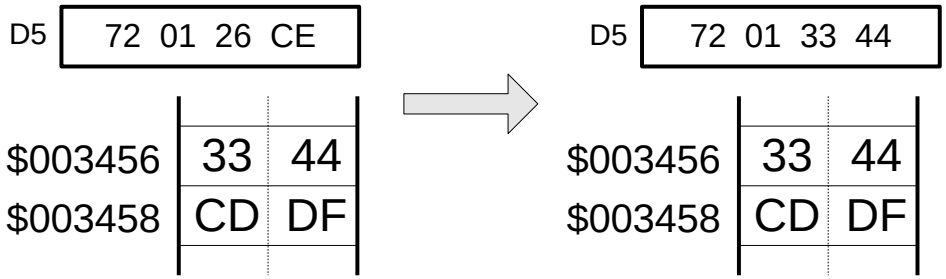
- **Example:**

Assembly Code

```
MOVE.W $3456,D5
```

Machine Code

```
$3A38 3456
```



MOVE Instruction

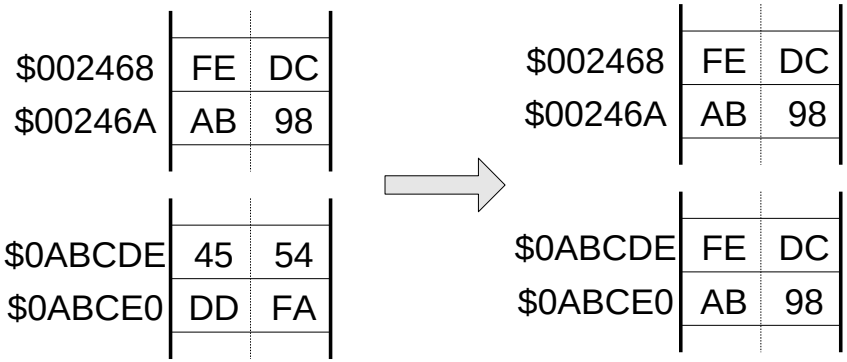
- **Example:**

Assembly Code

```
MOVE.L $2468,$ABCDE
```

Machine Code

```
$23F8 2468
$000A BCDE
```



MOVE Instruction

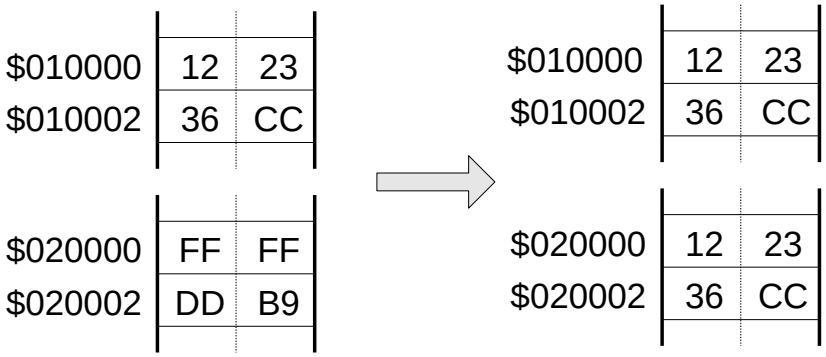
- **Example:**

Assembly Code

```
MOVE.L $10000,$20000
```

Machine Code

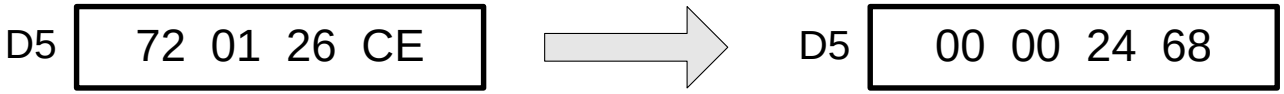
```
$23F9 0001
0000 0002 0000
```



Immediate Mode

- **Example:**

Assembly Code	Machine Code
`MOVE.L #$2468,D5`	`$2A3C 0000 2468`



Immediate Mode

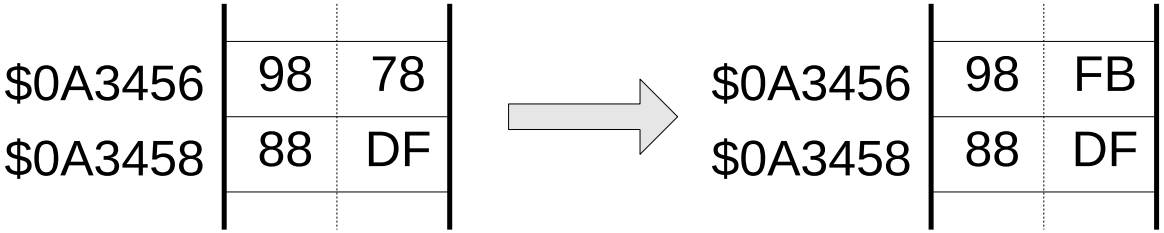
- **Example:**

Assembly Code

```
MOVE.B #-5,$20001
```

Machine Code

```
$13FC FFFB 000A 3457
```



Immediate Mode

Example (Illegal operation)

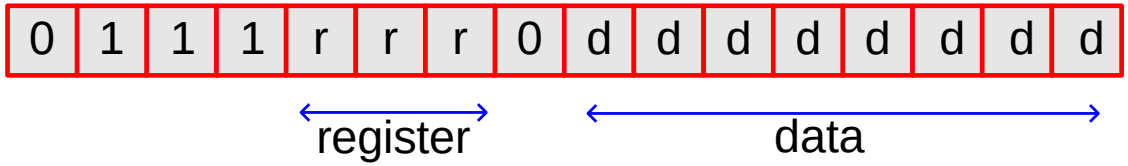
- `MOVE.B  #$2468,D2` → *Size of source is word*
- `MOVE.W  D2,#1234` → *Immediate mode can't be a destination*
- `MOVE.L  #1234,$ABCDEF` → *Longword must begin at even address*
- `CLR.B  #$ABCD` → *Immediate mode can't be a destination*

MOVEQ Instruction

- **Operation**: Fill constant in data register
- **Syntax** :

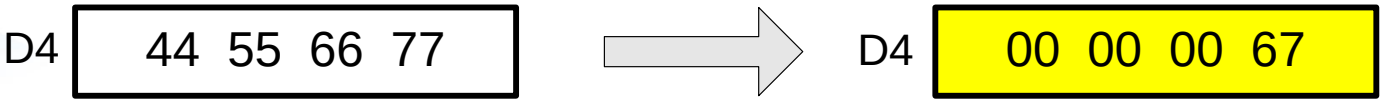
```
MOVEQ       #<data>,Dn
```

- **Advantage**: One word instruction and fast execution time (4cc)
- **Requirement**
  - Destination must be data register (`Dn`)
  - The value is 8-bit signed integer
  - No size require with sign extended format as all the content of selected data register are changed

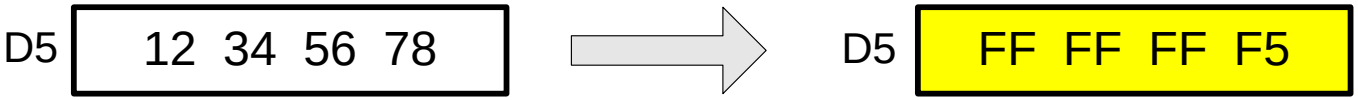


MOVEQ Instruction

- **Example**: `MOVEQ  #$67, D4`



- **Example**: `MOVEQ  -#11, D5`

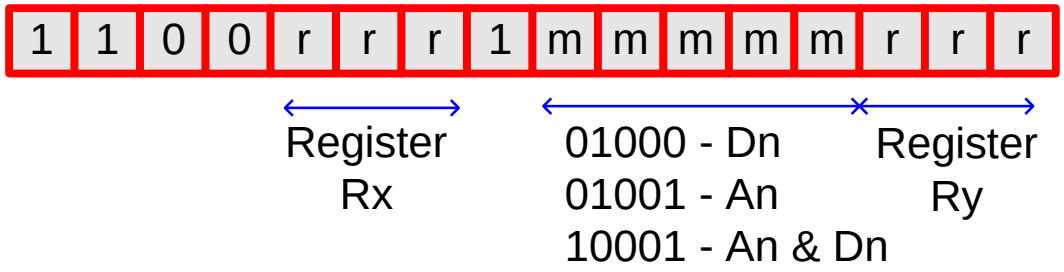


EXG Instruction

- **Operation**: Exchange the content of two registers (address or data register)
- **Syntax**

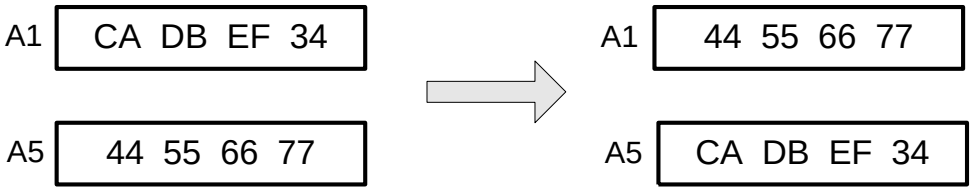
```
EXG   Rx,Ry
```

- **Requirement**
  - Longword operation (size operation is not needed)
  - Address register and data register only

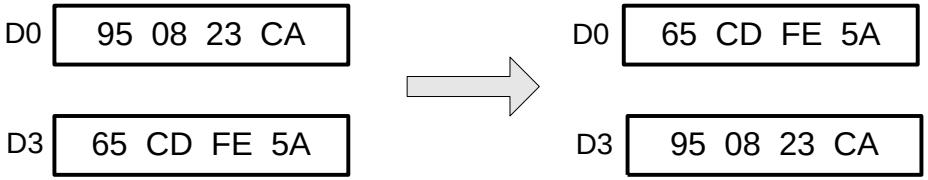


EXG Instruction

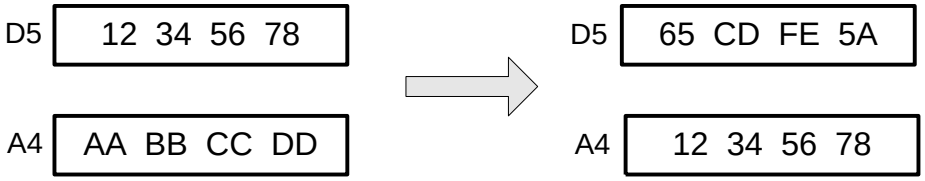
- **Example**: `EXG A1, A5`



- **Example**: `EXG D0, D3`



- **Example**: `EXG D5, A4`

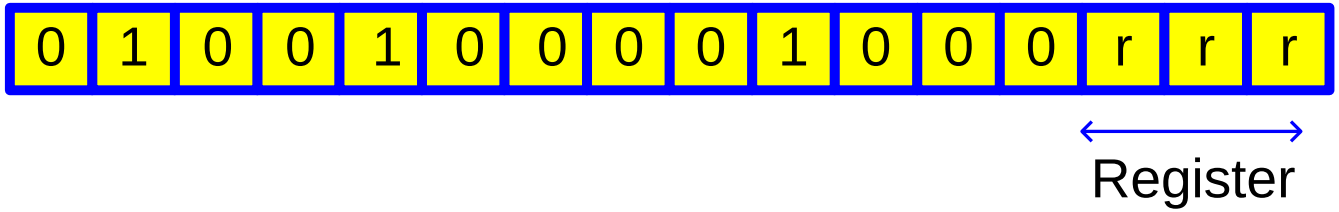


SWAP Instruction

- **Operation**: Exchange the upper word and lower word of data register
- **Syntax**

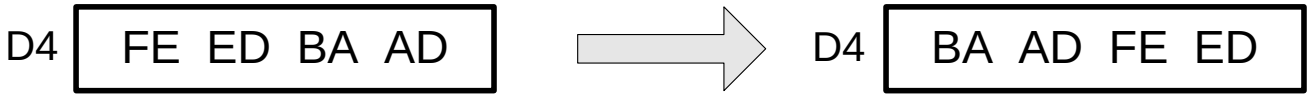
```
SWAP Dn
```

- **Requirement**
  - Data register only
  - No size required

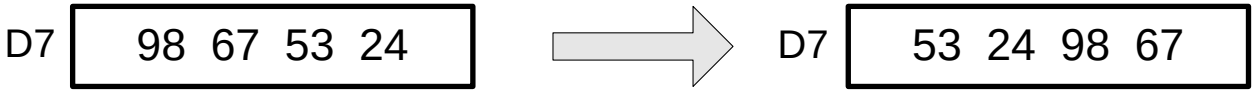


SWAP Instruction

- **Example**: `SWAP D4`



- **Example**: `SWAP D7`



Execution Time

- 68000 execution times are normally represented by clock cycle (cc)
- It is assumed that both memory read and write cycle times are four clock period
- The execution times depends on following factor
  - **Type of operation**
  - **Size of operation**
  - **Addressing mode for both operands**
- In order to obtain the execution time in unit second, the value of crystal must be determined
- For example, if CPU is operated under a 4MHz crystal, the `CLR.B D0` instruction is executed in 1 microsecond

More Examples..

- The content of registers

Register	D0	D1	D2	D3	D4	D5	D6
	12345678	246811AB	ABCD1111	11111010	AABBDD11	73425231	99110111

```
ORG         $1000
CLR.B       D2
MOVE.L      D1, D0
SWAP        D3
EXG         D5,D4
MOVEQ       #-2,D6
```

- The content of registers after the execution of the program

Register	D0	D1	D2	D3	D4	D5	D6
	246811AB	246811AB	ABCD1100	10101111	73425231	AABBDD11	FFFFFFFE

More Examples..

- The content of registers before execution

Register	PC	D1	D2	D3	D4	D5
	12345678	33334567	ABCDFEED	6464DEDA	4354ADCE	12345678

```
ORG         $2000
CLR.L       D1
MOVE.L      #20,D2
MOVEQ       #0,D3
MOVE.L      D5,$10000
MOVE.B      #$20,D4
```

- The content of registers after the execution of the program

Register	PC	D1	D2	D3	D4	D5
	00002014	00000000	00000014	00000000	4354AD20	12345678

More Examples..

- Obtain the machine code for each instruction

Instruction	Machine Code($)
CLR.L D1	4281
CLR.B D0	4200
CLR.W D5	4245
MOVE.W D6,$6000	31C6 6000
MOVE.L #-15,D4	283C FFFF FFF1
MOVE.L D2,D3	2602
MOVEQ #1,D7	7E01
EXG A0, D3	C788
SWAP D7	4847

More Examples..

- Calculate execution time and byte requirement for each instruction if CPU is operating at 1 MHz

Instruction	Time (μs)	# of bytes
CLR.L $10000	28	6
CLR.W $2000	16	4
MOVE.W #100,$20000	20	8
SWAP D4	4	2
MOVE.W $100,$200	20	6
MOVE.W #1,(A0)+	12	4
MOVEQ #1,D7	4	2
EXG A0, D3	6	2

Self-Test

- Calculate execution time and byte requirement for each instruction if CPU is operating at 10 MHz

Instruction	Time (μs)	# of bytes
CLR.L D5		
MOVE.L #1564, $2000		
ADD.B #45,D4		
MOVE.B (A0),$1000		
SUBQ.W #2,D7		
MULU D2,D3		
CLR.L 10(A0,D0)		
EXG D3,D4		

Self-Test

- Assemble the following instruction

Instruction	Machine Code($)
MOVEQ #-20,D4	
CLR.L 1234	
SWAP D1	
MOVE.B $100,$200	
MOVE.L #-150,(A0)	
EXG D3, D7	
MOVE.W D6,$5000	
ADD.B D0,D1	
CLR.B D1	

Self-Test

- The content of registers before execution

Register	PC	D1	D2	D3	D4	D5
	2345612	12345678	FEEDABCD	10101964	10002000	97864733

```
ORG         $6000
CLR.W       D5
MOVE.W      #200, D4
MOVEQ       #100,D2
SWAP        D2,D3
MOVE.L      #-@200,D1
```

- Obtain the content of registers and program memory map after the above program is assembled and executed

Self-Test

- What is the content of D2 and D3 if the following program is executed ?

```
MOVE.L #$12345678,D2
MOVE.L #$1ABCD980,D3
SWAP D3
EXG D2,D3
```

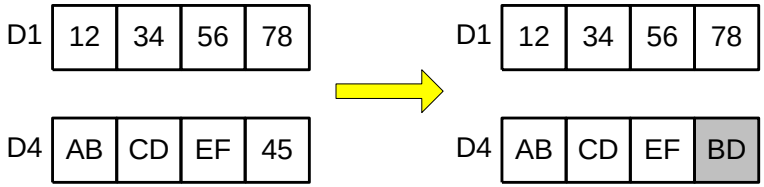
ADD Instruction

- **Operation** : To Add value to a destination and store the result in the destination
- **Requirement**
  - Destination + Source →Destination
  - One of the operands must be data register
  - All CCR are changed based on result
- **Format instruction set**

```
ADD.s      source,destination
```

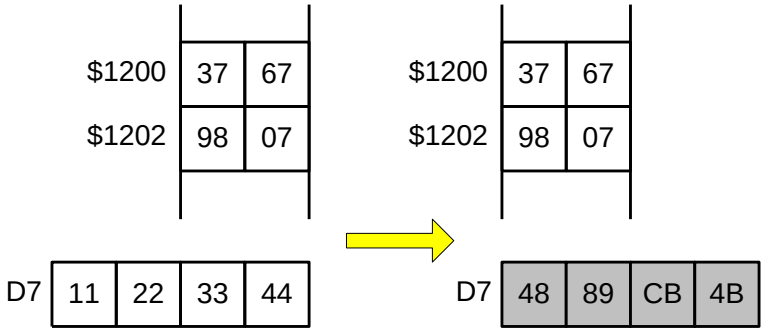
ADD Instruction

- **Example:** `ADD.B D1,D4`



- **Example:** `ADD.W #$12D4,D7`

- **Example:** `ADD.L $1200,D7`



ADD Instruction

- `ADD.W $2000,$3000` → *One of the operand must be Dn*

  Replaced by

```
MOVE.W $2000,D0
ADD.W D0,$3000
```

- `ADD.W D5,#$3000` → *Immediate mode cannot be destination*

- `ADD.B #140,D4` → *Size exceed the limit*

ADD - Version

- There are various version of ADD instructions
  - **ADDI.s #data,destination**
    - The source is immediate mode; that is data
  - **ADDQ.s #data,destination**
    - The source is immediate mode with value of 1-8
  - **ADDA.s Effective Address,An**
    - Destination is address register with size either word or longword
    - The content of CCR is unchanged
    - All content of address register are changed

CCR Register

- CCR produces the result of arithmetic operation
- Each bit is called flag and it is important in Bcc instruction
  - **Bit 0, Carry (C)**
    Set if the operation produce carry or borrow
  - **Bit1, Overflow (V)**
    Useful in sign magnitude operation
    Set if the add/subtract produces result not within the range
  - **Bit 2, Zero (Z)**
    Set if the result of operation produces zero
  - **Bit 3, Negative**
    Set if the result of operation produces negative value
  - **Bit 4, Extend (X)**
    Function only multiple word operation

ADD & CCR

- Example: `ADD.W D0,D1`

D0 before ADD	D1 before ADD	D1 after ADD	X N Z V C
0000 3040	0000 0002	0000 3042	0 0 0 0 0
0000 0000	0000 0000	0000 0000	0 0 1 0 0
0000 3040	0000 CFC0	0000 0000	1 0 1 0 1
0000 8000	0000 8000	0000 0000	1 0 1 1 1
0000 FF00	0000 0002	0000 FF02	0 1 0 0 0
0000 7000	0000 7000	0000 E000	0 1 0 1 0

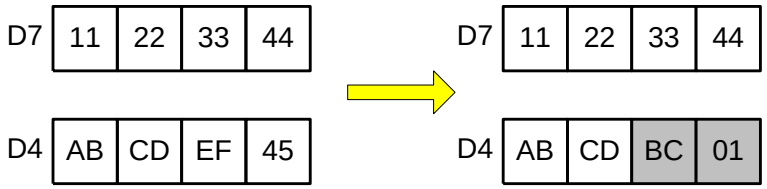
SUB Instruction

- **Operation** : To subtract value from a destination and store the result in the destination
- **Requirement**
  - Destination - Source →Destination
  - One of the operands must be data register
  - All CCR are changed based on result
- **Format instruction set**

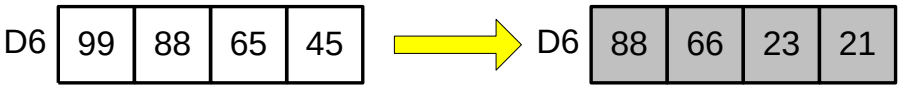
```
SUB.s      source,destination
```

SUB Instruction

- **Example:** `SUB.W D7,D4`



- **Example:** `SUB.L #$11224224,D6`



- **Example:** `SUB.W #$2222,$3222` → **Error !!** Replaced by

```
MOVE.W $3222,D0
SUB.W #$2222,D0
```

SUB - Version

- There are various version of SUB instructions
  - **SUBI.s #data,destination**
    - The source is immediate mode; that is data
  - **SUBQ.s #data,destination**
    - The source is immediate mode with value of 1-8
  - **SUBA.s Effective Address,An**
    - Destination is address register with size either word or longword
    - The content of CCR is unchanged
    - All content of address registers are changed

SUB & CCR

- **Example:** `SUB.W D2,D3`

D0 before SUB	D1 before SUB	D1 after SUB	X N Z V C
0000 3040	0000 0002	0000 303E	0 0 0 0 0
0000 0040	0000 0040	0000 0000	0 0 1 0 0
0000 FFFF	0000 55AA	0000 AA55	0 1 0 0 0
0000 0000	0000 0300	0000 FD00	1 1 0 0 1
0000 9000	0000 7000	0000 2000	0 0 0 1 0
0000 7000	0000 9000	0000 E000	1 1 0 1 1

Integer Multiplication

- There are two type of integer multiplications
  - Multiply for signed number

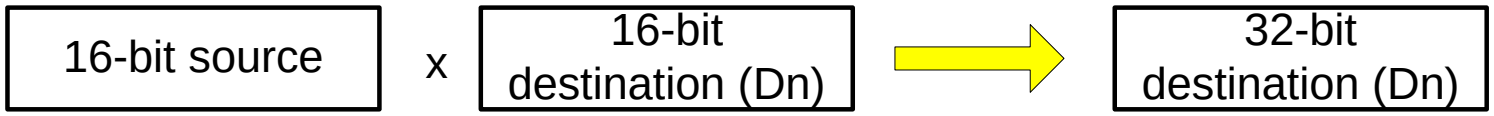
```
MULS    source,Dn
```

  - Multiply for unsigned number

```
MULU    source,Dn
```

Integer Multiplication

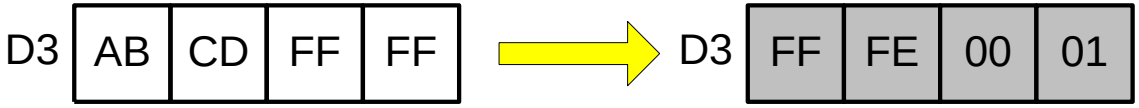
- **Requirement**
  - **Source** must be **16 bits**
  - **Destination** must be **16 bits Data Register**
  - Product of multiplication is stored in destination with the size of **32 bits**



Integer Multiplication

- **Example:** `MULU #3,D2`

- **Example:** `MULU #$FFFF,D3`



- **Example:** `MULS #$FFFF,D1`

Integer Division

- There are two type of integer divisions
  - Divide for signed number

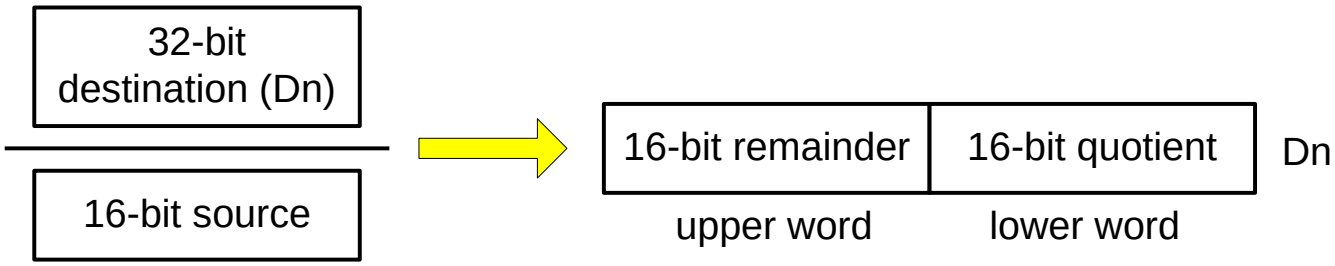
```
DIVS    source,Dn
```

  - Divide for unsigned number

```
DIVU    source,Dn
```

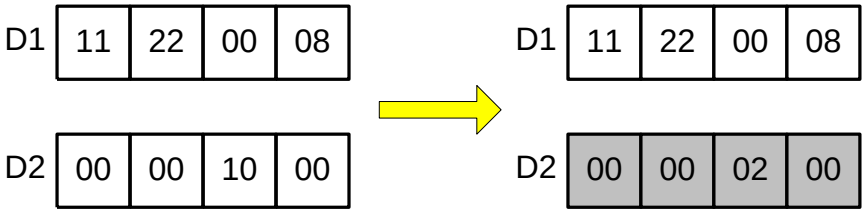
Integer Division

- **Requirement**
  - Source is **denominator** and must be **16 bits**
  - Destination is **numerator** and must be **32 bits Data Register**
  - Result of division is stored in **Data Register** with **quotient in lower word** and **remainder in upper word**



Integer Division

- **Example:** `DIVU D1,D2`



- **Example:**

```
MOVE.L #$40000,D6
DIVU #2,D6
```

→ **Error ? Why ?**

- **Example:**

```
MOVE.L #4569,D1
CLR.L D4
DIVU D4,D1
```

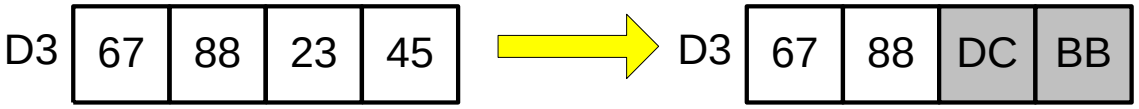
→ **Error ? Why ?**

NEG Instruction

- **Operation**: Produce the negative number of the content of data register (Dn) or obtain the 2's complement of the content Dn

- **Example**: `NEG.B D2`

- **Example**: `NEG.L D3`



EXT Instruction

- **Operation:** To extend the sign of the content
- There are two type

```
EXT.W   Dn
```

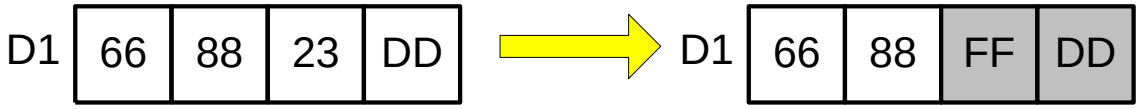
- To extend sign from byte to word
  Sign bit (bit 7) is duplicated to all upper byte

```
EXT.L   Dn
```

- To extend sign from word to longword
  Sign bit (bit 15) is duplicated to all upper word

EXT Instruction

- **Example :** `EXT.W D1`



- **Example:** `EXT.L D4`

More Examples..

- The size of two values to be added must be the same
- **Example :** Add a byte in D0 with a word in D1(signed number)

```
EXT.W D0
ADD.W D0,D1
```

- **Example** : Add a byte in D4 with a longword in D5 (signed number)

```
EXT.W D4
EXT.L D4
ADD.L D4,D5
```

More Examples..

- **Example :** Add a byte in D0 with a word in D1 (unsigned number)

```
CLR.W   D2
MOVE.B  D0,D2
ADD.W   D2,D1
```

- **Example :** Add a word in D2 with a longword in D3 (unsigned number)

```
CLR.L  D4
MOVE.W  D2,D4
ADD.L  D4,D3
```

More Examples..

- **Example** : Add a byte in D4 with a longword in D5 (unsigned)

```
CLR.L D6
MOVE.B D4,D6
ADD.L D6,D5
```

- **Example** : Divide a 16-bit number stored in location $1000 by another located at location $1002. Store the result in location $1004

```
MOVE.L    #0,D0
MOVE.W    $1000,D0
DIVU      $1002,D0
MOVE.W    D0,$1004
```

More Examples..

- **Example** : Square a byte-sized number stored in memory location $3000. Store the result in location $3002

```
MOVE.L#0,D1
MOVE.B$3000,D1
MULU D1,D1
MOVE.L D1,$3002
```

More Examples..

- **Example** : Find the average of 5 16-bit numbers stored in location $2000 - $2008. Store the average value in location $2010

```
MOVEQ #0,D0
ADD.W $2000,D0
ADD.W $2002,D0
ADD.W $2004,D0
ADD.W $2006,D0
ADD.W $2008,D0
DIVU #5,D0
MOVE.W D0,$2010
```

Self-Test

Exercise

If D0=$12345678 and D1=$87654321, evaluate the following instruction and obtain the new value of D0 and D1

Instruction	D0	D1
ADD.L D0,D1		
SUBI.W #$3245,D0		
MULU D0,D1		
DIVS #$32,D1		
EXT.L D0		
EXT.W D1		
NEG.L D0		

Self-Test

Exercise

If D4=$02468ABCD, D5=$ABCDFEED and SR=$0010, evaluate the following instruction and obtain the new value of D4,D5 and SR

Instruction	D4	D5	SR
ADD.B D4,D5			
SUB.W D5,D4			
MULS #$FFFE,D5			
DIVU #8,D4			
ADDI.W #$8765,D5			
EXT.W D5			
NEG.W D4			

Self-Test

Exercise

State the syntax error for each instruction

Instruction	Syntax Error
`ADDQ.B #10,$1000`	
`SUBI.L D1,D5`	
`MULS D2,$2000`	
`EXT.B D7`	
`ADD.W $100,$200`	
`NEG.B $200`	
`DIV #$12345,D6`	

Self-Test

Exercise

If D3=$4545FEDC, D4=$FEBA3423 and D2=$12348765, obtain the content of each register if the following program is executed

```
MOVE.W #-16,D2
SWAP   D3
EXT.L   D4
EXG D2,D3
MULS D2,D2
NEG.L D2
```

Shift & Rotate Instruction

- There are 8 instructions that can be used to shift or rotate the operand
  - `ASL` (Arithmetic Shift Left)
  - `ASR` (Arithmetic Shift Right)
  - `LSL` (Logical Shift Left)
  - `LSR` (Logical Shift Right)
  - `ROL` (Rotate Left)
  - `ROR` (Rotate Right)
  - `ROLX` (Rotate Left through X)
  - `RORX` (Rotate Right through X)

Shift & Rotate Instruction

- Syntax for operation

```
Operation.s Dx,Dy
```

- Register Dy is n times shifted determined by Dx (only lowest 6 bits)

```
Operation.s #data,Dy
```

- Register Dn is shifted by n times (1<n<7)

```
Operation.W <ea>
```

- Shifted one bit to the left

Arithmetic Shift

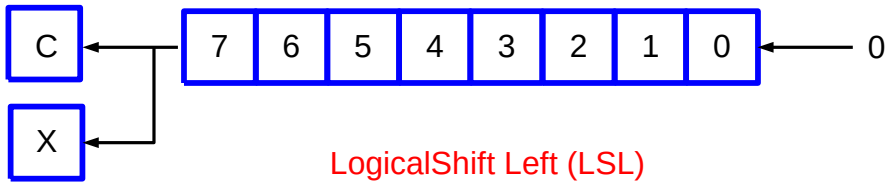
- The value to be shifted is assumed to be sign number
- Instruction ASL shifts the operand to the left with each shift equivalent to multiply by two
- Instruction ASR shifts the operand to the right with each shift equivalent to division by two

Arithmetic Shift Left
(ASL)

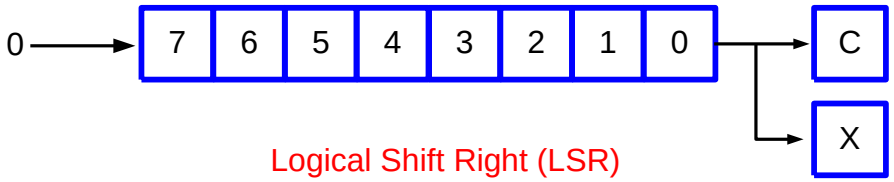
Arithmetic Shift Right
(ASR)

Logical Shift

- The value to be shifted is assumed to be unsigned number
- Instruction LSL shifts the operand to the left with each shift equivalent to multiply by two
- Instruction LSR shifts the operand to the right with each shift equivalent to division by two



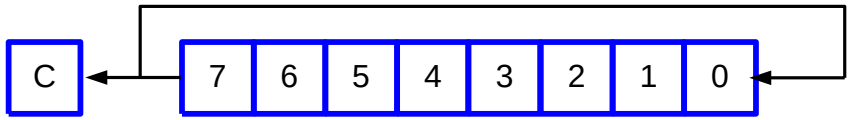
LogicalShift Left (LSL)



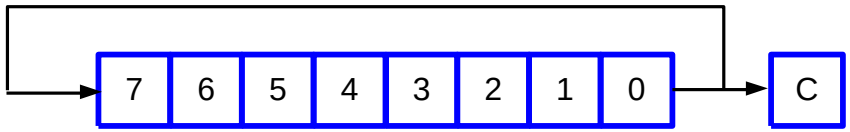
Logical Shift Right (LSR)

Rotate

- `ROL` instruction rotates the operand to the left with the MSB is fed to LSB and carry bit
- `ROR` instruction rotates the operand to the right with the LSB is fed to MSB and carry bit



Rotate Left
(ROL)



Rotate Right
(ROR)

More Examples..

- **Example**: Evaluate each instruction and obtain the content of each register or memory

Instruction	Before	After
ASL.W #6,D0	D0=$12345678 CCR=%00100	D0=$12349E00 CCR=%11011
ASR.L D0,D1	D0=$10005FCE D1=$A2345678 CCR = %11011	D0=$10005FCE D1=FFFE88E CCR=%01000
LSR $2000	($2000)=$1234 CCR=%11010	($2000)=091A CCR=%00000
ROR.W #8,D2	D2=$12345678 CCR=%10001	D2=$12347856 CCR=%10000

Bit Manipulation

- Bit Manipulation tests a single bit in destination
- Only byte and word operation are allowed
- Only zero flag and selective bit are changed
- There are four types

- **Bit Change**

```
BCHG Dn,<ea>
BCHG #data,<ea>
```

- **Bit Clear**

```
BCLR Dn,<ea>
BCLR #data,<ea>
```

- **Bit Set**

```
BSET   Dn,<ea>
BSET   #data,<ea>
```

- **Bit Test** (only zero flag change)

```
BCLR   Dn,<ea>
BCLR   #data,<ea>
```

More Example..

- **Example**: Obtain the content of register or memory if the following instruction is executed

```
A0=$00002000,D0=$1F04A215,D1=$214A2271,
($2000)=$8BA6
```

Instruction	After Execution
BTST.B #2,$2000	($2000)=$8BA6,Z=1
BCLR D0,1(A0)	($2001)=$86,Z=0
BSET D1,D0	D0=$1F06A215,Z=1
BCHG #29,D1	D1=$014A2271,Z=0

More Example..

- **Example 1**

  If address $200000 contains $FD and instruction `BTST.B #4,$200000` is executed, the content of $200000 remain the same but the zero flag is clear (Z=0)

- **Example 2**

  D0 contains $ABCDEFFE. If instruction `BCLR.L #$29,D0`, D0 will be $8BCDEFFE and the flag is clear (Z=0)

More Example..

- **Example 3**

  D5 has $FEDD1234 and instruction `BCHG.L #7,D5` is executed. The D5 then has FEDD9234 with Z is set

- **Example 4**

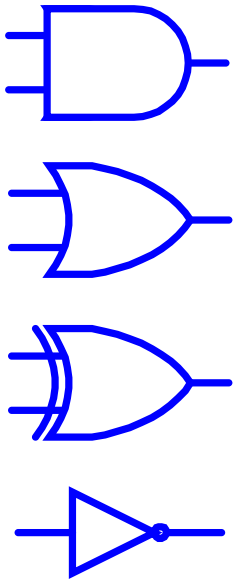
  If location $1234 has $CE and instruction `BSET.B #6,$1234` is executed, the location $1234 remains the same and the flag is clear (Z=0)

Boolean Instruction

- The syntax for the operation

```
Operation.s #data,<dea>
```

```
Operation.s <sea>,<dea>
```



- There are four logical instructions
  - **AND** - To clear more than one bit in an operand
  - **OR** - To set more than one bit in an operand
  - **EOR** - To change more than one bit in an operand
  - **NOT** - To complement all bits in an operand

More Examples..

- **Example**: Evaluate each instruction and obtain the content of each register or memory

Instruction	Before	After
AND.B (A0),D4	D4=$12345678 (A0)=$F034	D4=$12345630 (A0)=$F034
ORI #$F0F0,(A0)	(A0)=$5216	(A0)=$F2F6
EOR.W D2,$2000	D2=0000FFFF ($2000)=$5678	D2=0000FFFF ($2000)=$A987
NOT.B D2	D2=$8124659	D2=$81246566

More Examples..

- **Example 1 :** Clear all bit except bit 28-31 in D1

```
ANDI.L #$F0000000,D1
```

- **Example 2 :** Set all even bits in D2

```
ORI.L    #$55555555,D2
```

More Examples..

- **Example 3 :** Change all bits in lower word of D3

```
EORI.L #$FFFF,D3
```

or

```
NOT.W    D3
```

- **Example 4 :** Write an equivalent instruction for `BSET.L #30,D3`

```
ORI.L #$40000000,D3
```

Self-Test

Exercise

If D2=$ABCD3412, D4=$87654321 and SR=$0000, evaluate the following instruction and obtain the new value of D2, D4 and SR

Instruction	D2	D4	SR
ASL.L #4,D2			
LSR.B #5,D4			
ROL.L #8,D2			
ASR.L D2,D4			
AND.L D4,D2			
NOT.W D2			
EORI.B #$AA,D4			

Self-Test

- **Exercise** : Write a program to set parity bit of a byte that located in D0
- **Exercise**: Evaluate the following instruction if

```
D0=$11224215,D1=$ACDE3271,A0=$00002000
($2000)=8DA6 CE24
```

Instruction	D0	D1	Z-flag
`BTST.B #2,$2000`			
`BCLR.B D0,1(A0)`			
`BSET.L D1,D0`			
`BCHG.L #29,D1`			
`ORI.L D0,D1`			