CSE-3103: Microprocessor and Microcontroller

Dept. of Computer Science and Engineering
University of Dhaka

Prof. Sazzad M.S. Imran, PhD
Dept. of Electrical and Electronic Engineering
sazzadmsi.webnode.com

Arithmetic Instructions

AAM →

ASCII Adjust for Multiplication.
Adjusts result of multiplication of 2 unpacked BCD values = pair of unpacked BCD values.
Multiplication should not be performed in ASCII.
AX register = source and destination operand.

AAM is only useful when it follows
MUL instruction = multiplies 2 unpacked BCD values and
AX ← word result.
AAM adjusts contents of AX = correct 2-digit unpacked BCD result.

Example 1:

```
mov     AL, 3           ; multiplier in unpacked BCD form [0011]
mov     BL, 9           ; multiplicand in unpacked BCD form [1001]
mul     BL              ; result 001BH is in AX [0001 1011]
aam                     ; AX := 0207H
```

Arithmetic Instructions

AAM →

Example 1:

```
        aam                 ; AX := 0207H [result = 001BH]
                            ; B > 9, so add 6 to it → B+6 = 17H.
                            ; LSD of 17H = lower unpacked byte for result.
                            ; Increment AH by 1, 1+1 = 2 =
                            ; upper unpacked byte of result.
                            ; After execution, AH = 02 and AL = 07.
        or      AX, 3030H   ; AX := 3237H
```

Example 2:

```
        mov     AL, '3'     ; multiplier in ASCII
        mov     BL, '9'     ; multiplicand in ASCII
        and     AL, 0FH     ; multiplier in unpacked BCD form
        and     BL, 0FH     ; multiplicand in unpacked BCD form
        mul     BL          ; result 001BH is in AX
        aam                 ; AX := 0207H
        or      AX, 3030H   ; AX := 3237H
```

Arithmetic Instructions

AAD →

ASCII Adjust for Division [= ASCII adjust before division].
Converts 2 unpacked BCD digits in AL and AH → binary number in AL.
Adjustment is made before
2 unpacked BCD digits in AX ÷ unpacked BCD byte.
After AAD →
AL ← (AH×0Ah) + AL
AH ← 00H
SF, PF, ZF are modified.
CF, AF, OF are not defined.

Assume AX = 0508 = unpacked BCD for 58 decimal, and DH = 02H.
AAD → hexadecimal 3A in AL and 00 in AH.

	AH	AL	
AX	05	08	05×0Ah + 8 = 58D = 3AH
AAD	00	3A	05×0Ah = 50D = 32H + 8 = 3AH

Arithmetic Instructions

AAD →

After AAD →

AL ← (AH×0Ah) + AL

AH ← 00H

Example:

divide 27 by 5

```
mov     AX, 0207H       ; dividend in unpacked BCD form
mov     BL, 05H         ; divisor in unpacked BCD form
aad                     ; AX := 001BH [AL = 2×0Ah = 14H + 7 = 1BH]
div     BL              ; AX := 0205H
```

Arithmetic Instructions

CBW →

Converts signed byte to signed word.
Extends sign bit of AL into AH register.

AL ← Byte to be converted.
AX = Result and preserves number's sign.
Does not affect any flag.

Example:

```
byte_val  SBYTE  -101    ; -101 = -65h = 9Bh = 1001 1011
mov       al, byte_val   ; AL = 9Bh
cbw                      ; AX = FF9Bh
```

Arithmetic Instructions

CWD →

Extends sign bit of AX into DX register.
This operation is to be done before signed division.
It does not affect any flag.

Example:

```
word_val  SWORD  -101          ; FF9Bh
mov       ax, word_val         ; AX = FF9Bh
cwd                            ; DX:AX = FFFFh:FF9Bh
```

Logic Instructions

AND →

Performs bitwise AND operation.
Bitwise AND operation =
returns 1, if matching bits from both operands are 1,
returns 0, otherwise.

Example:
Operand1 = 0101 and Operand2 = 0011
AND Operand1, Operand2
→ Operand1 = 0001

Operand1 or destination = register or memory,
Operand2 or source = register, memory or immediate value.
Both source and destination cannot be memory in single instruction.

Logic Instructions

OR →

Performs bitwise OR operation.
Bitwise OR operation =

returns 1, if matching bits from either or both operands are 1.
returns 0, if both bits are 0.

Example:

operand1 = 0101 and operand2 = 0011
OR operand1, operand2
→ operand1 = 0111

operand1 or destination = register or memory,
operand2 or source = register, memory or immediate value.
Both source and destination cannot be memory in single instruction.

Logic Instructions

XOR →

Implements bitwise XOR operation.
bits from both operands are different →
resultant bit = 1.
bits from both operands are same →
(= both 0 or both 1)
resultant bit = 0.

Example:
Operand1 = 0101 and Operand2 = 0011
XOR Operand1, Operand2
→ Operand1 = 0110

operand1 or destination = register or memory,
operand2 or source = register, memory or immediate value.
Both source and destination cannot be memory in single instruction.

XORing operand with itself → operand = 0.
used to clear register.

Logic Instructions

NOT →

Implements bitwise NOT operation.
NOT operation reverses bits in operand.
Operand = register or memory.

Example:

operand1 = 0101 0011
NOT operand1
→ operand1 = 1010 1100

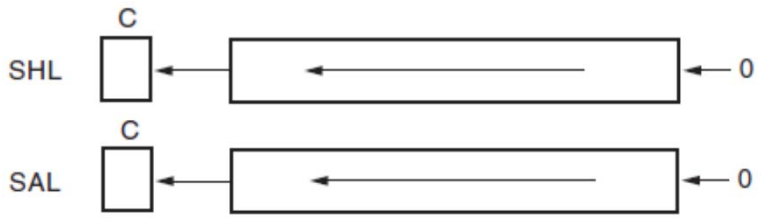
Shift Instructions

Position or move numbers to left or right.
Numbers = register or memory location.

Arithmetic left shift =
multiplication by powers of 2^n
Arithmetic right shift =
division by powers of 2^n

Microprocessor's instruction set = 4 different shift instructions.

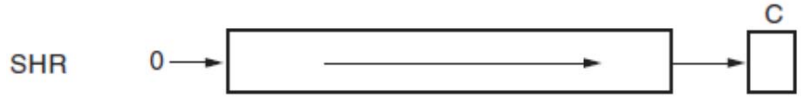
SAL/SHL D, Count →
shift D left by Count number of bit positions,
fill vacated bit positions on right with zeros.



Shift Instructions

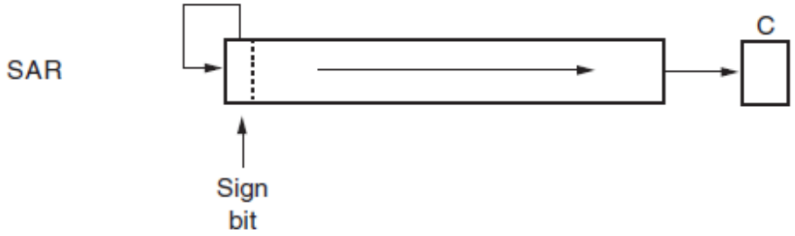
SHR D, Count →

shift D right by Count number of bit positions,
fill vacated bit positions on left with zeros.



SAR D, Count →

shift D right by Count number of bit positions,
fill vacated bit positions on left with original MSB.



Shift Instructions

Example →

```
; multiply AX by 10 (1010)
        SHL     AX, 1           ; AX times 2
        MOV     BX, AX
        SHL     AX, 2           ; AX times 8
        ADD     AX, BX          ; AX times 10

; multiply AX by 18 (10010)
        SHL     AX, 1           ; AX times 2
        MOV     BX, AX
        SHL     AX, 3           ; AX times 16
        ADD     AX, BX          ; AX times 18

; multiply AX by 5 (101)
        MOV     BX, AX
        SHL     AX, 2           ; AX times 4
        ADD     AX, BX          ; AX times 5
```

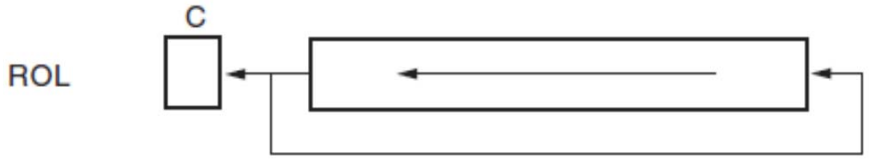
```
        MOV     DX, AX
        SHL     AX, 1
        SHL     DX, 3
        ADD     AX, DX
```

```
        MOV     BX, AX
        SHL     AX, 1
        SHL     BX, 4
        ADD     AX, BX
```

Rotate Instructions

ROL r/m, op1 →

shifts each bit in register or memory operand specified to left.
MSB is copied into CF and into LSB position.
No bits are lost.



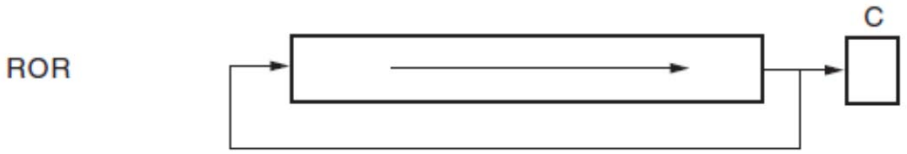
Example:

```
mov     al, 11110000b
rol     al, 1               ; AL = 11100001b, CF = 1
```

Rotate Instructions

ROR r/m, op1 →

shifts each bit in register or memory operand specified to right,
copies LSB into CF and into MSB position.
No bits are lost.



Example:

```
mov     dl, 3Fh         ; DL = 00111111b
ror     dl, 4           ; DL = 11110011b = F3h, CF = 1
```

1st shift →
DL = 10011111b, CF = 1
2nd shift →
DL = 11001111b, CF = 1
3rd shift →
DL = 11100111b, CF = 1
4th shift →
DL = 11110011b, CF = 1

Rotate Instructions

RCL r/m, op1 →

shifts each bit in register or memory operand specified to left,
copies CF to LSB,
copies MSB into CF.

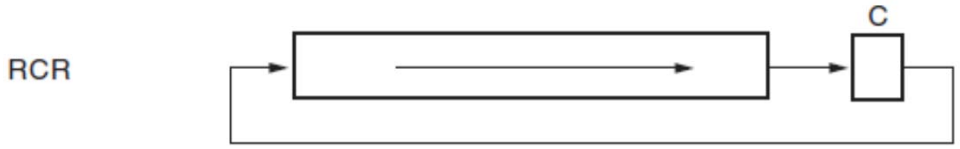
Example:

```
clc                 ; clear carry, CF = 0
mov     bl, 88h     ; CF = 0, BL = 10001000b
rcl     bl, 1       ; CF = 1, BL = 00010000b
rcl     bl, 1       ; CF = 0, BL = 00100001b
```

Rotate Instructions

RCR r/m, op1 →

shifts each bit in register or memory operand to right,
copies CF into MSB,
copies LSB into CF.



Example:

```
stc                 ; set carry, CF = 1
mov     ah, 10h     ; CF = 1, AH = 00010000b
rcr     ah, 1       ; CF = 0, AH = 10001000b
```

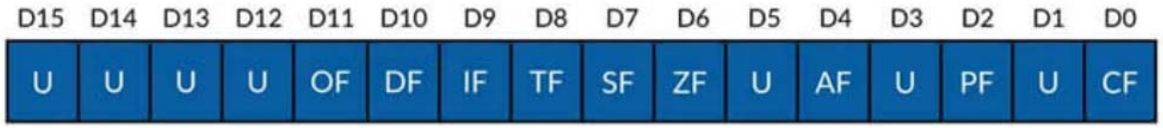
Flags Control Instructions

Modify some of flag bits of 8086.
Control functioning of hardware inside processor chip.

LAHF →

Load AH from flags.
Transfers low byte of flags word to AH register.
Bits (lsb to msb) are →

sign, zero, indet, auxiliary carry, indet, parity, indet, carry.



SAHF →

Store AH into flags.
Loads flags (SF, ZF, indet, AF, indet, PF, indet, CF) with values.

CMC →

Complementary carry flag.
Reverses setting of carry flag;
Affects no other flags.

Flags Control Instructions

CLC →

Clear carry flag.
Sets carry flag to zero;
Affects no other flags.

STC →

Sets carry flag to 1.
Affects no other flags.

CLI →

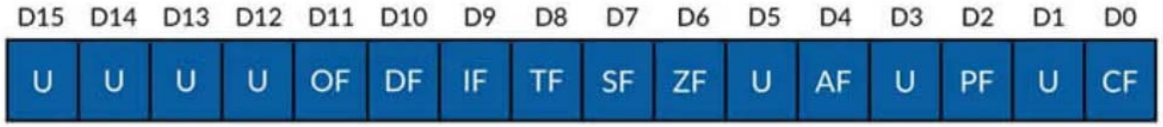
Clear interrupt flag (IF = 0).
Affects no other flags.
External interrupts disabled until IF = 1.

STI →

Sets interrupt flag to 1.
Affects no other flags.

CLD →

Clear direction flag.
Affects no other flags or registers.
All subsequent string operations →
increment index registers.



STD →

Set direction flag to 1,
All subsequent string operations →
decrement index registers.

CMP Instructions

Characteristics of CMP instruction →

(i) Can compare two 8-bit or two 16-bit numbers.

(ii) Operands may reside in memory, register or be part of instruction.

(iii) Results of comparison is reflected in 6 status flags →

CF, AF, OF, PF, SF, ZF.

(iv) CMP = subtraction = uses 2's complement.

(v) Result of CMP is not saved.

Based on CMP result, appropriate flags are either set/reset.

CMP D, S →

Performs comparison between (D) and (S).

Comparison = signed subtraction of (S) from (D).

Appropriate status flag bits are updated.

Difference is then discarded.

(S) is immediate value →

sign extended to length of (D).