Lecture 5

1 8086 Instruction Set

1.1 Arithmetic Instructions

- 8086 arithmetic operations can be performed on four types of numbers: unsigned binary, signed binary, unsigned packed decimal, and signed packed decimal numbers

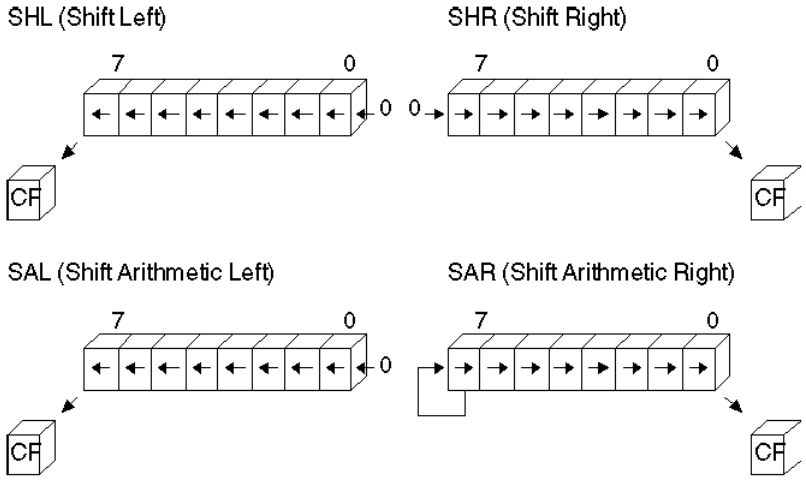
1.1.1 Additiona and Subtraction

```
add     dest, src; dest := dest + src
adc     dest, src; dest := dest + src + C
SUB     dest, src; dest := dest   src
sbb     dest, src; dest := dest - src   C
neg     dest     ; dest := - dest
inc     dest     ; dest := dest + 1
dec     dest     ; dest := dest   1
```

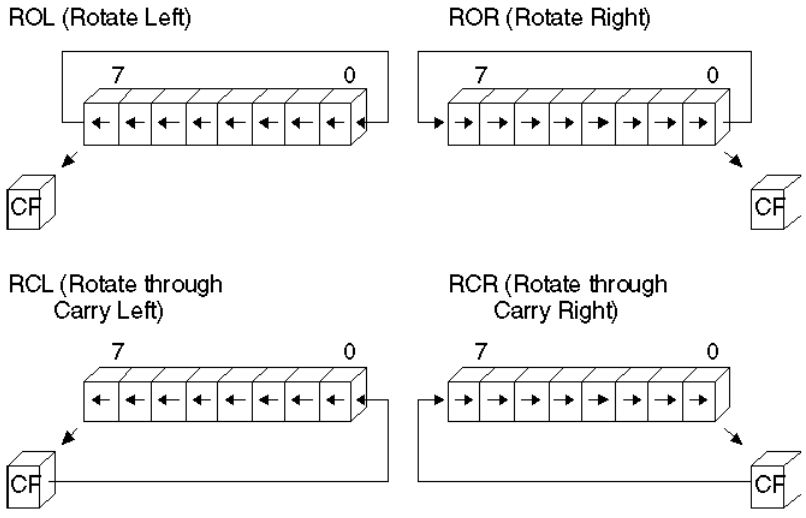
- Src and dest can be any combination of a register, a memory location, or an immediate data
- Dest has to be a register or a memory location
- Both Dest and Src can NOT be memory locations.
- These instructions would affect the following flags
  - The overflow flag denotes a **signed** arithmetic overflow.
  - The carry flag denotes an **unsigned** arithmetic overflow.
  - The sign flag denotes a negative result (i.e., the H.O. bit of the result is one).
  - The zero flag is set if the result is zero.
  - The auxiliary carry flag contains one if a BCD overflow out of the L.O. nibble occurs.
  - The parity flag is set or cleared depending on the parity of the result
- **Inc** is faster than add reg,1.

1.2 Bit Manipulation Instructions

- Operations include logicals, shifts, and rotates
- operand can be either 8- or 16-bit
- Logical operations include AND, OR, XOR, NOT
  - AND can be used for ***masking*** and checking zero flag
  - OR can be used to ***insert a 1*** in a particular bit position
  - Syntax: AND dest,src
- Shift Instructions include SHL/SAL, SHR, SAR
  - Syntax: SHL dest,Count
  - SHL/SHR: last bit shifted out goes to CF and 0 shifted in



- Rotate instructions include ROL, ROR, RCR, RCL
  - Syntax: ROL mem/reg,count



1.3 Interrupt Instruction (Software Interrupts)

- calls a DOS interrupt (like a function) to perform a special task such as read from the keyboard or disk or mouse, or write to the screen
- Format: INT n
  - n refers to a the kind of the interrupt.
- In DOS, there are a over 50 different interrupts available
- Interrupts may have a number of **sub-functions** which select the individual task that the function has to do

1.3.1 INT 21h

Subfunction	Input	output
AH = 01h - READ CHARACTER FROM STANDARD INPUT		Return: AL=character read
AH = 02h -WRITE CHARACTER TO STANDARD OUTPUT	DL = character to write	Return: AL = last character output
AH = 05h - WRITE CHARACTER TO PRINTER	DL = character to print	
AH = 09h - WRITE STRING TO STANDARD OUTPUT	DX=address of string	Return: AL = 24h
AH = 0Ah - BUFFERED INPUT		Return: buffer filled with user input

vspace12pt

```
1 MOV AH,02 ; Function to output a char
2 MOV DL,"!" ; Character to output
3 INT 21h ; Call the interrurpt to output "!"
4 MOV AH,04Ch ; Select exit function
5 MOV AL,00 ; Return 0
6 INT 21h ; Call the interrupt to exit
```

```
mov dx, msg ; load offset of msg into dx
mov ah, 09h ; print function is 9.
int 21h ; do it!
mov ah, 0
int 16h ; wait for any key....
ret ; return to operating system.
msg: db "Hello, World!", 0Dh,0Ah, 24h
```

1.4 Stack-related Instructions

1.4.1 Push and POP

- PUSH and POP are respectively used to write data to and read data from the stack
- Stack works as LIFO (Last In First Out) system
  - SP is decremented by 2 for PUSH
  - SP is incremented by 2 for POP
- Example: (DS)=2000**H**; (BX)=0200**H**; (SP)=3000**H**; (SS)=4000**H**; and (20200) = 0120**H**
  - Push BX
  - PUSH [BX]
- 8086 also has the following commands
  - PUSHA → pushes all registers
  - PUSHF → pushes flag register
  - POPA → do the reverse of PUSHA
  - POPF → do the reverse of PUSHF

Example

1.5 Compare Instruction

- Format: cmp dest, src ;dest - src (and set flags)
- CMP is identical to the SUB instruction but it does not store the result to the destination
- Flags can be checked for specific conditions

1.6 Iteration Control Instructions

- LOOP disp;
  - decrements the CX register by 1 without affecting the flags and performs JMP if CX <> zero otherwise proceed.
- LOOPE (Loop while equal → ZF=1) / LOOPZ (Loop while zero) goes to the label if condition is satisfied.
- LOOPNE/LOOPNZ disp;
- JCXZ disp; JMP if register CX =0

1.7 Conditional Branch Instructions

- The conditions are typically checked after arithmetic and logical operation.
- Conditional branch instructions use 8-bit signed displacement (-128 to +127,)
- Compare signed numbers: JG, JE, JGE, JLE, JE, and JNE
- Compare unsigned numbers: JA, JB, JAE, JBE, JE, and JNE
- Flag Branch: JC, JNC, JP, JNP, JO, JNO, JS, JNS, JZ, JNZ
- Example that checks for pressing ESC

```
start: mov ah,01; sub-function 1
       int 21h ; read the character
       CMP al,27; compare it to the escape character
       JNZ start ; if it is not equal then go to escape
       mov ah,02; subfunction 2 - output a character.
       mov dl,"E"; output letter "E"
       int 21h
       mov dl,"S"; then output "S".
       int 21h
       mov dl,"C"; finally output "C"
       int 21h
       RET  ; if any other key is pressed execution continues here
```

- Example: find the number of matches for an 8-bit number in an 8086 register such as DL in a data array of 50 bytes

```
       MOV AL, 0; Clear AL to 0, AL to hold number of matches
       MOV CX,15; Initialize array count
       Mov BX, 0;
START: CMP DL, mynumbers[BX] ; Compare the number to be matched in DL
                             ;with a data byte in the array.
       JZ MATCH; If there is a match, ZF=l. Branch to MATCH.
       JMP DOWN; Unconditional jump to label DOWN.
MATCH: INC AL; increment AL to hold number of matches.
DOWN:  INC BX; Increment BX to point to next data byte.
       LOOP START  ;  Decrement CX by 1, go back to START if
                   ; CX <>0. If CX=0, go to the next instruction
                   ; AL   contains the number of matches
  hlt
mynumbers DB 10,13,0, 0, 14,16, 5, 3,0, 9, 0, 12, 34, 9, 0
```

```
; calculate the sum of elements in vector, store result in m and print it in bi
start: mov cx, 5; number of elements
mov al, 0; al will store the sum
```

```
mov bx, 0; bx is an index
next: add al, vector[bx]; sum elements
inc bx; next byte:
loop next; loop until cx=0:
mov m, al; store result in m:
mov bl, m; print result in binary:
mov cx, 8; number of bits per byte
print:mov ah, 2    ; print function.
                  mov dl, '0'; prepare for printing zero
                  test bl, 10000000b  ; test first bit.
                  jz zero
                  mov dl, '1'; prepare for printing zero
zero:    int 21h
                  shl bl, 1
                  loop print; loop back for next bit
                  mov dl, 'b'; print binary suffix:
                  int 21h
                  mov ah, 0; wait for any key press:
                  int 16h
                  ret;

vector db 3, 4, 5, 2, 1; variables:
m db 0
```

References

M. RAFIQUZZAMAN, "Fundamentals of Digital Logic and Microcomputer Design," Fifth Edition.

Appendix: Assembly Statements

- Assembly has three classes of statements
- executable instructions: each statement typically generates one machine language instruction
  - assembler directives or pseudo-ops: statements provides information to the assembler on various aspects of the assembly process
  - Macros: shorthand notation for a group of statements that provide the basic input and output capabilities to standalone assembly language programs
- Assembly language statements are entered one per line in the source file.
- Assembly statement format
  - each statement can have up to **four fields**
    - Label field [optional]

- an identifier: appears in an executable instruction and is used as a marker to identify the instruction. Points to the memory address of the instruction.
- a constant used with certain assembler directives such as EQU.
- Labels and other names can be formed from upper and lowercase letters (a to z, A to Z), digits (0 through 9), and special characters (_, %, ?, $, ., @).
- A label may not begin with a digit
- Aperiod is only allowed at the beginning
- Examples of legal names

```
COUNTER1
@character
SUM_OF_DIGITS
$1000
DONE?
.TEST
```

- Examples of illegal names

```
TWO WORDS contains a blank
2abc begins with a digit
A45.28 . not first character
YOU&ME contains an illegal character
```

* Instruction field
  - a required field and identifies the purpose of the statement
  - Examples are: ADD, MOV, SUB
* Operand field [optional]
  - specify the data to be manipulated by the statement and its number depends on the instruction
  - instructions may have zero or more operands
  - operands are typically separted by comma ","
  - Examples

```
NOP ;no operands -- does nothing
INC AX ;one operand -- adds 1 to the contents of AX
ADD WORD1,2 ;two operands -- adds 2 to the contents
```

* Comment field [optional]
  - begin with a semicolon (;) and extend until the end of the line.

Label	Mnemonic	Operand	Comment
Start:	MOV	AX,01H	;move 1 to AX

– The fields in a statement must be separated by at least one space or tab character.