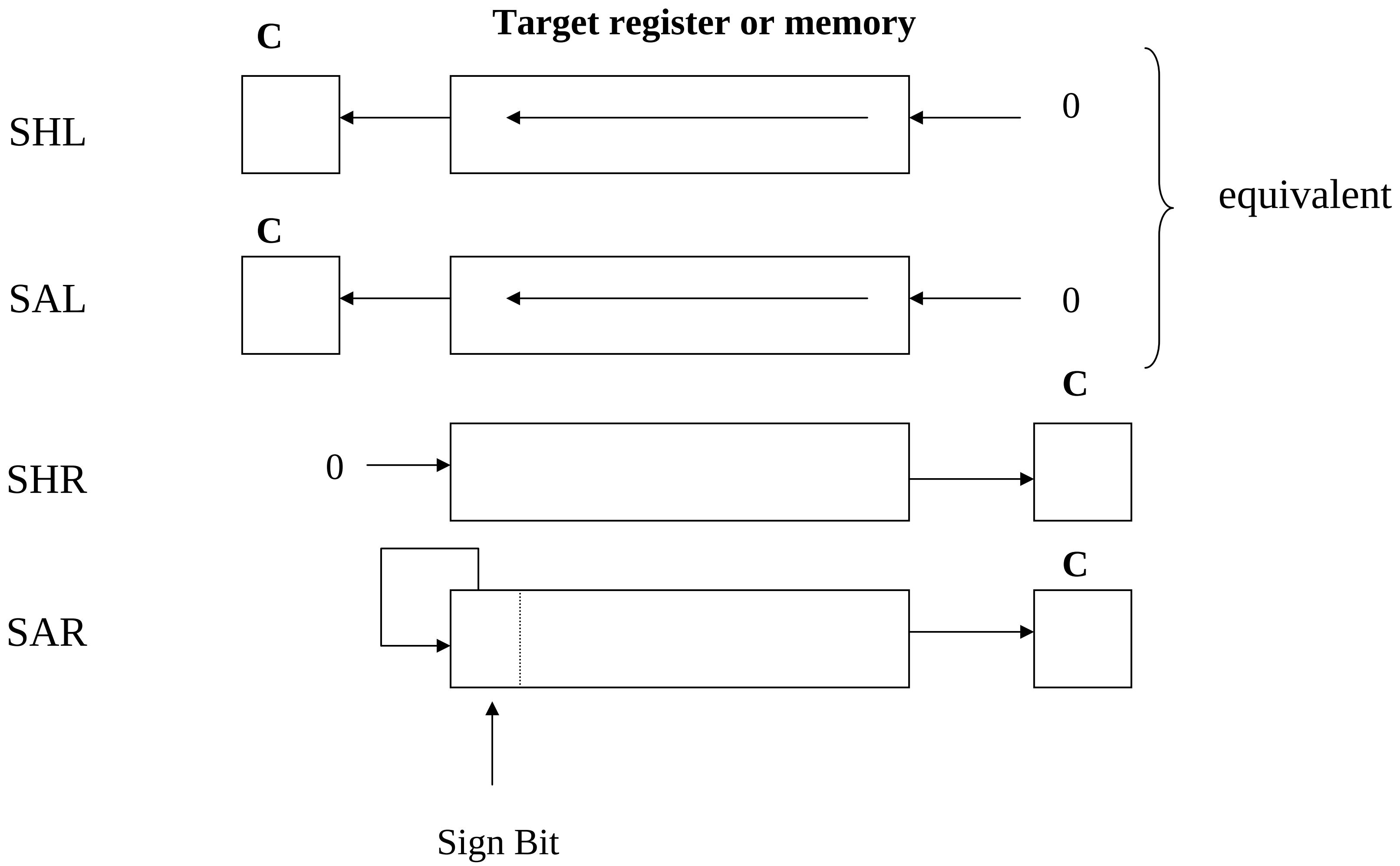
Weeks 6

8088/8086 Microprocessor Programming

Shift



Target register or memory
C
SHL
0
equivalent
C
SAL
0
C
SHR
0
C
SAR
Sign Bit

Examples

Examples

```
SHL AX,1
SAL DATA1, CL   ; shift count is a modulo-32 count
```

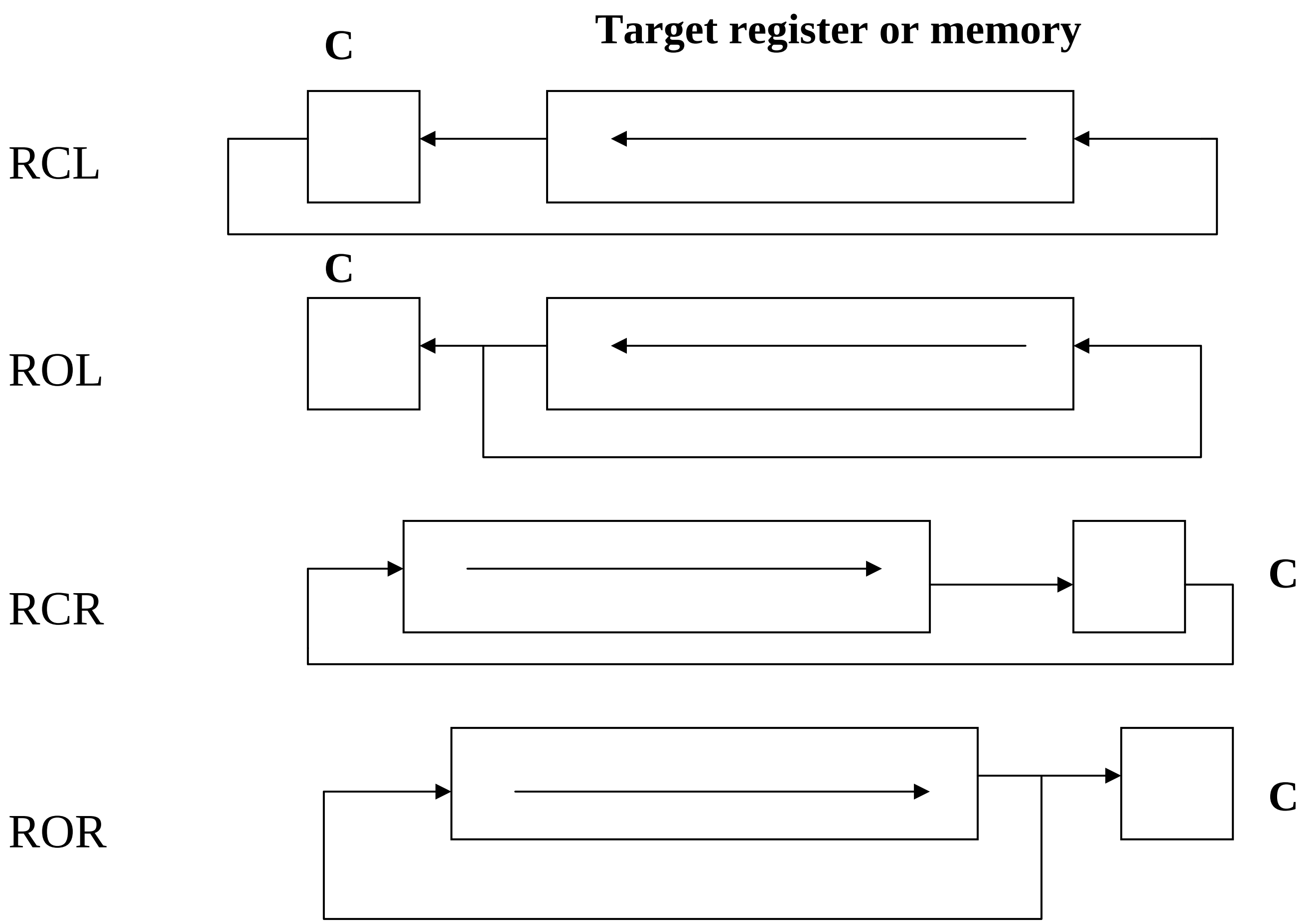
Ex.

```
; Multiply AX by 10
        SHL AX, 1
        MOV BX, AX
        MOV CL,2
        SHL AX,CL
        ADD AX, BX
```

Ex. What are the results of SAR CL, 1 if CL initially contains B6H?

Ex. What are the results of SHL AL, CL if AL contains 75H and CL contains 3?

Rotate



Ex.

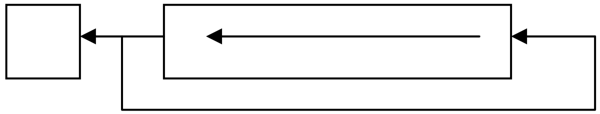
What is the result of ROL byte ptr [SI], 1 if this memory location 3C020 contains 41H?

What is the result of ROL word ptr [SI], 8 if this memory location 3C020 contains 4125H?

Example

Write a program that counts the number of 1's in a byte and writes it into BL

```
DATA1   DB 97                  ; 61h
        SUB    BL,BL           ;clear BL to keep the number of 1s
        MOV    DL,8            ;rotate total of 8 times
        MOV    AL,DATA1
AGAIN:  ROL    AL,1            ;rotate it once
        JNC    NEXT            ;check for 1
        INC    BL              ;if CF=1 then add one to count
NEXT:   DEC    DL              ;go through this 8 times
        JNZ    AGAIN           ;if not finished go back
        NOP
```



BCD and ASCII Numbers

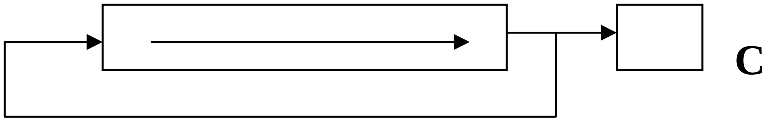
- BCD (Binary Coded Decimal)
  - Unpacked BCD: One byte per digit
  - Packed BCD: 4 bits per digit (more efficient in storing data)
- ASCII to unpacked BCD conversion
  - Keyboards, printers, and monitors all use ASCII.
  - Digits 0 to 9 are represented by ASCII codes 30 – 39.

- Example. Write an 8086 program that displays the packed BCD number in register AL on the system video monitor
  - The first number to be displayed should be the MS Nibble
  - It is found by masking the LS Nibble and then rotating the MS Nibble into the LSD position
  - The result is then converted to ASCII by adding 30h
  - The BIOS video service is then called to display this result.

ASCII Numbers Example

```
MOV BL,AL; save
AND AL,F0H
MOV CL,4
ROR AL,CL
ADD AL,30H
MOV AH,0EH
INT 10H ;display single character

MOV AL,BL; use again
AND AL,0FH
ADD AL,30H
INT 10H
INT 20H             ; RETURN TO DOS
```



Example

- Write an 8086 program that adds two packed BCD numbers input from the keyboard and computes and displays the result on the system video monitor
- Data should be in the form 64+89= The answer 153 should appear in the next line.

#	?	6	4	+	8	9	=
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

Example Continued

```
Mov dx, offset bufferaddress
Mov ah,0a
Mov si,dx
Mov byte ptr [si], 6
Int 21
Mov ah,0eh
Mov al,0ah
Int 10
; BIOS service 0e line feed  position cursor

 sub byte ptr[si+2], 30h
 sub byte ptr[si+3], 30h
 sub byte ptr[si+5], 30h
 sub byte ptr[si+6], 30h
```

6	?	6	4	+	8	9	=
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

```
Mov cl,4
Rol byte ptr [si+3],cl
Rol byte ptr [si+6],cl
Ror word ptr [si+5], cl
Ror word ptr [si+2], cl

 Mov al, [si+3]
 Add al, [si+6]
 Daa
 Mov bh,al
 Jnc display
 Mov al,1
 Call display
 Mov al,bh
 Call display
 Int 20
```

Flag Control Instructions

SF	ZF		AF		PF		CF

- **LAHF** Load AH from flags (AH) $\leftarrow$ (Flags)
- **SAHF** Store AH into flags (Flags) $\leftarrow$ (AH)
  - Flags affected: SF, ZF, AF, PF, CF

Bulk manipulation of the flags

- **CLC** Clear Carry Flag (CF) $\leftarrow$ 0
- **STC** Set Carry Flag (CF) $\leftarrow$ 1
- **CLI** Clear Interrupt Flag (IF) $\leftarrow$ 0
- **STI** Set interrupt flag (IF) $\leftarrow$ 1

Individual manipulation of the flags

- <u>Example (try with debug)</u>

```
LAHF
MOV AX,0000
ADD AX,00
SAHF
```

  - Check the flag changes!

Compare

Mnemonic	Meaning	Format	Operation	Flags Affected
CMP	Compare	CMP D,S	(D) − (S) is used in setting or resetting the flags	CF, AF, OF, PF, SF, ZF

(a)

Unsigned Comparison		
Comp Operands	CF	ZF
Dest > source	0	0
Dest = source	0	1
Dest < source	1	0

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

(b)

Signed Comparison		
Comp Operands	ZF	SF,OF
Dest > source	0	SF=OF
Dest = source	1	x
Dest < source	0	SF<>OF

Compare Example

```
DATA1        DW     235Fh
…
MOV AX, CCCCH
CMP AX, DATA1
JNC OVER
SUB AX,AX
OVER: INC DATA1
```

CCCC – 235F = A96D => Z=0, CF=0 =>
CCCC > DATA1

Compare (CMP)

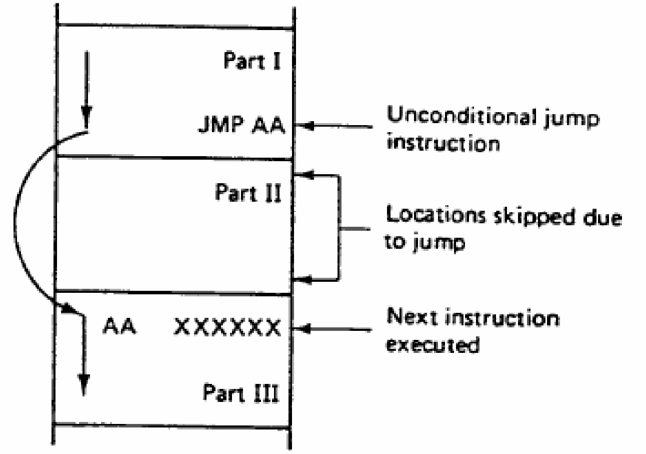
For ex: CMP CL,BL ; CL-BL; no modification on neither operands

Write a program to find the **highest** among 5 grades and write it in **DL**

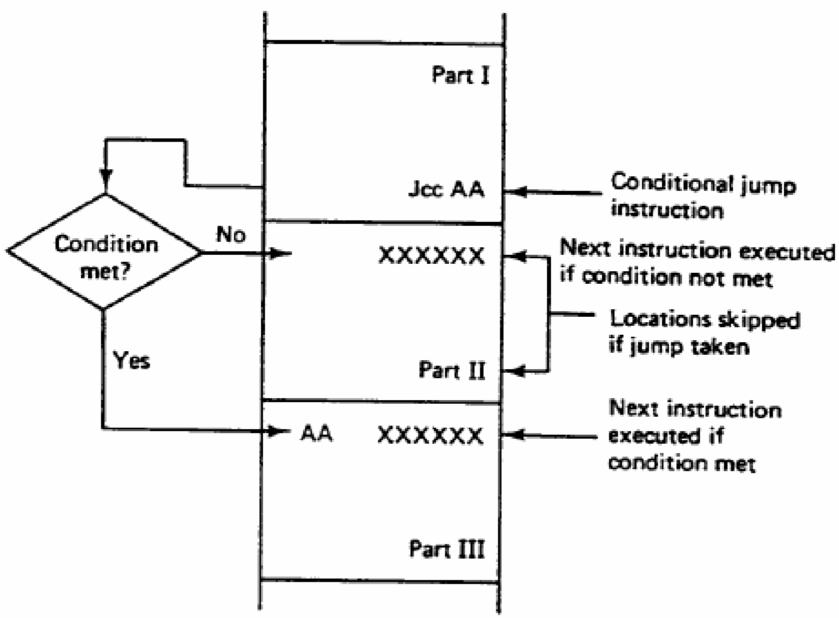
```
DATA    DB      51, 44, 99, 88, 80          ;13h,2ch,63h,58h,50h
        MOV     CX,5                        ;set up loop counter
        MOV     BX, OFFSET DATA             ;BX points to GRADE data
        SUB     AL,AL                       ;AL holds highest grade found so far
AGAIN:  CMP     AL,[BX]                     ;compare next grade to highest
        JA      NEXT                        ;jump if AL still highest
        MOV     AL,[BX]                     ;else AL holds new highest
NEXT:   INC     BX                          ;point to next grade
        LOOP AGAIN                          ;continue search
        MOV  DL, AL
```

Jump Instructions

- Unconditional vs conditional jump



(a)



(b)

Conditional Jump

These flags are based on general comparison

Mnemonic	Description	Flags/Registers
JZ	Jump if ZERO	ZF = 1
JE	Jump if EQUAL	ZF = 1
JNZ	Jump if NOT ZERO	ZF = 0
JNE	Jump if NOT EQUAL	ZF = 0
JC	Jump if CARRY	CF = 1
JNC	Jump if NO CARRY	CF = 0
JCXZ	Jump if CX = 0	CX = 0
JECXZ	Jump if ECX = 0	ECX = 0

Conditonal Jump based on flags

Mnemonic	Description	Flags/Registers
JS	JUMP IF SIGN (NEGATIVE)	SF = 1
JNS	JUMP IF NOT SIGN (POSITIVE)	SF = 0
JP	Jump if PARITY EVEN	PF = 1
JNP	Jump if PARITY ODD	PF = 0
JO	JUMP IF OVERFLOW	OF = 1
JNO	JUMP IF NO OVERFLOW	OF = 0

Jump Based on Unsigned Comparison

These flags are based on unsigned comparison

Mnemonic	Description	Flags/Registers
JA	Jump if above op1>op2	CF = 0 and ZF = 0
JNBE	Jump if not below or equal op1 not <= op2	CF = 0 and ZF = 0
JAE	Jump if above or equal op1>=op2	CF = 0
JNB	Jump if not below op1 not <opp2	CF = 0
JB	Jump if below op1<op2	CF = 1
JNAE	Jump if not above nor equal op1< op2	CF = 1
JBE	Jump if below or equal op1 <= op2	CF = 1 or ZF = 1
JNA	Jump if not above op1 <= op2	CF = 1 or ZF = 1

Jump Based on Signed Comparison

These flags are based on signed comparison

Mnemonic	Description	Flags/Registers
JG	Jump if GREATER op1>op2	SF = OF AND ZF = 0
JNLE	Jump if not LESS THAN or equal op1>op2	SF = OF AND ZF = 0
JGE	Jump if GREATER THAN or equal op1>=op2	SF = OF
JNL	Jump if not LESS THAN op1>=op2	SF = OF
JL	Jump if LESS THAN op1<op2	SF <> OF
JNGE	Jump if not GREATER THAN nor equal op1<op2	SF <> OF
JLE	Jump if LESS THAN or equal op1 <= op2	ZF = 1 OR SF <> OF
JNG	Jump if NOT GREATER THAN op1 <= op2	ZF = 1 OR SF <> OF

Control Transfer Instructions (conditional)

- It is often necessary to transfer the program execution.
  - Short
    - A special form of the direct jump: “short jump”
    - **All conditional jumps are short jumps**
    - Used whenever target address is in range +127 or –128 (single byte)
    - Instead of specifying the address a relative offset is used.

Short Jumps

- **Conditional Jump is a two byte instruction.**
- **In a jump backward the second byte is the 2's complement of the displacement value.**
- **To calculate the target the second byte is added to the IP of the instruction after the jump.**

Ex:

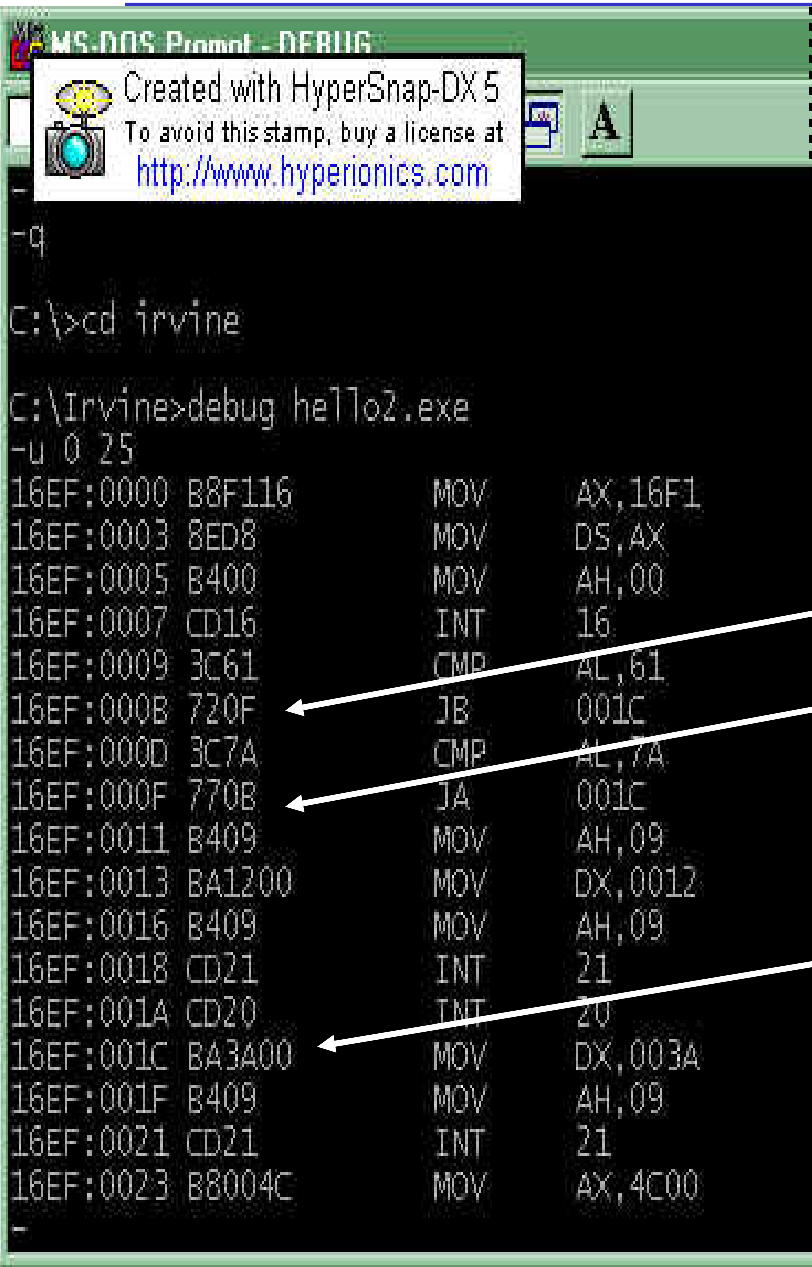
```
000D ADD AL,[BX]
000F INC BX
0010 DEC CX
0011 JNZ FA
0013
```

Short Jump 0013 + FA (-6)

= 0D

Hello2.exe

SJ Example




```
.model small
.stack 100h
.data
org 0010
message1 db "You now have a small letter
entered !",0dh,0ah,'$'
org 50
message2 db "You have NON small letters
",0dh,0ah,'$'
.code
    main proc
        mov ax,@data
        mov ds,ax
        mov ah,00h
        int 16h
        cmp al,61h
        jb next
        Cmp al,7Ah
        ja next
        mov ah,09h
        mov dx,offset message1
        mov ah,09h
        int 21h
        int 20h
        next: mov dx,offset message2
        mov ah,09h
        int 21h
        mov  ax,4C00h
        int  21h
    main endp
end main
```

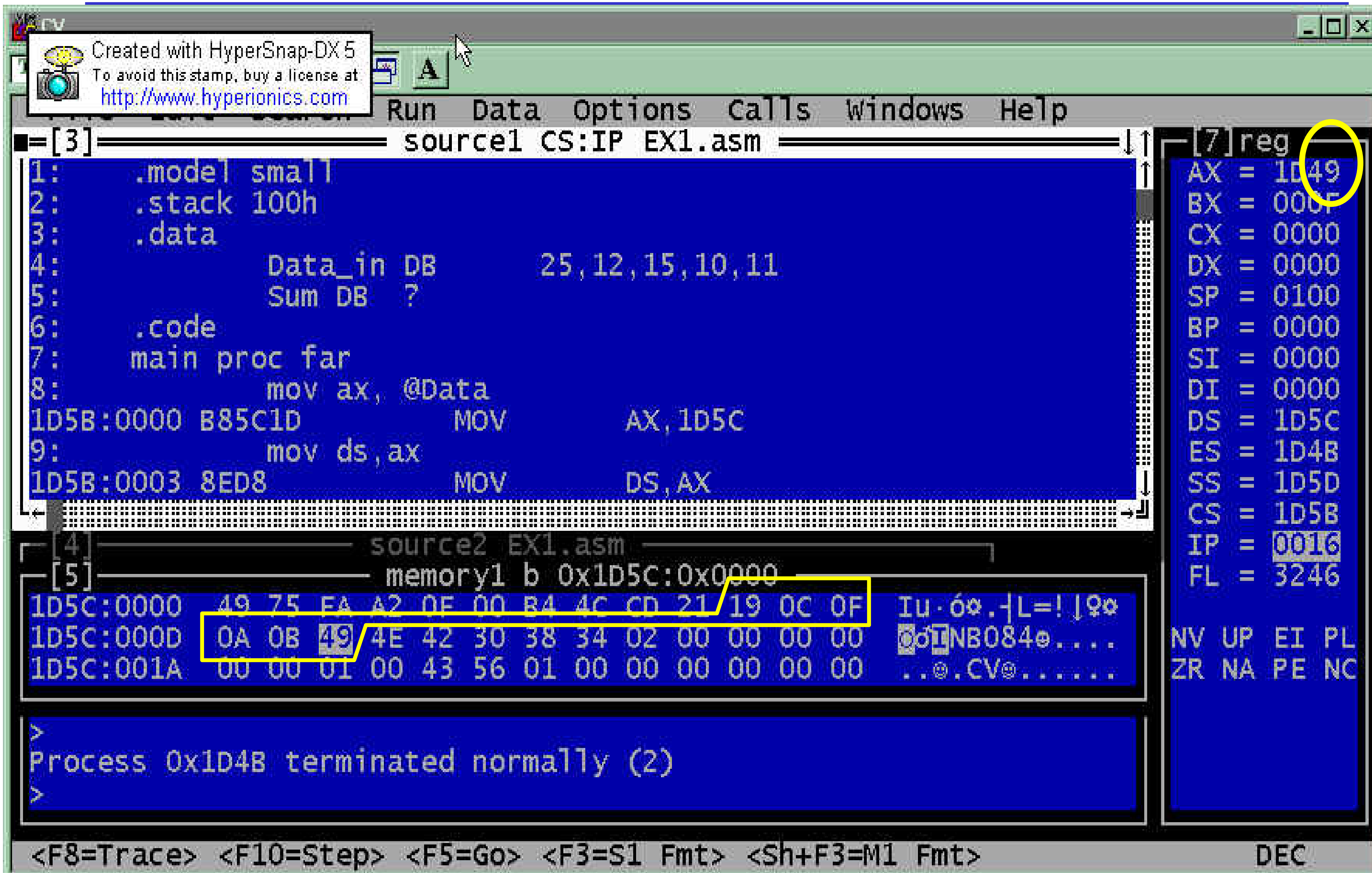
A Simple Example Program finds the sum

- Write a program that adds 5 bytes of data and saves the result. The data should be the following numbers: 25,12,15,10,11

```
.model small
.stack 100h
.data
        Data_in DB 25,12,15,10,11
        Sum DB  ?
.code
main proc far
        mov ax, @Data
        mov ds,ax
        mov cx,05h
        mov bx,offset data_in
        mov al,0
```

```
Again: add al,[bx]
       inc bx
       dec cx
       jnz Again
       mov sum,al
       mov ah,4Ch
       INT 21H
Main   endp
end main
```

Example Output



Unconditional Jump

❖Short Jump: jmp short L1 (8 bit)

❖Near Jump: jmp near ptr Label

If the control is transferred to a memory location within the current code segment (intrasegment), it is NEAR. IP is updated and CS remains the same

➢The displacement (16 bit) is added to the IP of the instruction following jump instruction. The displacement can be in the range of –32,768 to 32,768.

➢The target address can be register indirect, or assigned by the label.

➢**Register indirect JMP:** the target address is the contents of two memory locations pointed at by the register.

➢Ex: JMP [SI] will replace the IP with the contents of the memory locations pointed by DS:DI and DS:DI+1 or JMP [BP + SI + 1000] in SS

❖Far Jump: If the control is transferred to a memory location outside the current segment. Control is passing outside the current segment both CS and IP have to be updated to the new values. ex: JMP FAR PTR label = EA 00 10 00 20

jmp far ptr Label ; this is a jump out of the current segment.

Near Jump

```
0B20:1000 jmp 1200
0B20:1003
-u 1000
0B20:1000 E9FD01          JMP     1200
0B20:1003 200B            AND     [BP+DI],CL
```

Jumps to the specified IP with +/- 32K distance from the next instruction following the jmp instruction

Far Jump

```
0B20:1000 jmp 3000:1200
0B20:1005
-u 1000
0B20:1000 EA00120030      JMP     3000:1200
0B20:1005 FF750B          PUSH    [DI+0B]
```

Jumps to the specified CS:IP

XLAT

- Adds the contents of AL to BX and uses the resulting offset to point to an entry in an 8 bit translate table.
- This table contains values that are substituted for the original value in AL.
- The byte in the table entry pointed to by BX+AL is moved to AL.

- XLAT [tablename] ; optional because table is assumed at BX

- Table db ‘0123456789ABCDEF’

Mov AL,0A; index value

Mov bx,offset table

Xlat; AL=41h, or ‘A’

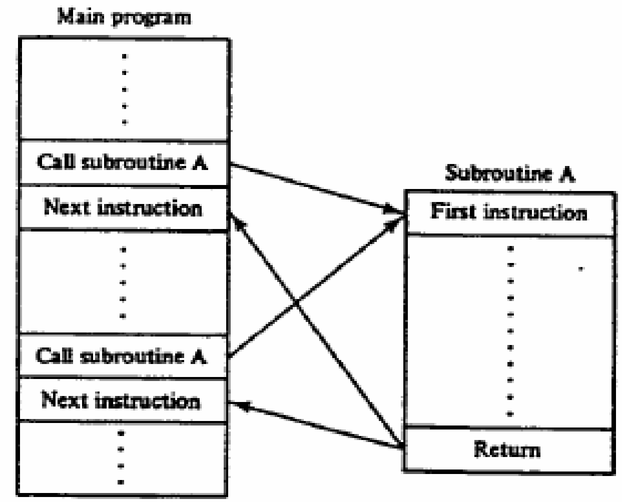
Subroutines and Subroutine Handling Functions

✓A subroutine is a special segment of a program that can be called for execution from any point in the program

✓A RET instruction must be included at the end of the subroutine to initiate the return sequence to the main program environment

Examples. **Call 1234h**
Call BX
Call [BX]

Two calls
- intrasegment
- intersegment



(a)

Mnemonic	Meaning	Format	Operation	Flags Affected
CALL	Subroutine call	CALL operand	Execution continues from the address of the subroutine specified by the operand. Information required to return back to the main program such as IP and CS are saved on the stack.	None

(b)

Operand
Near-proc
Far-proc
Memptr16
Regptr16
Memptr32

(c)

Figure 6–20 (a) Subroutine concept. (b) Subroutine call instruction. (c) Allowed operands.

Calling a NEAR proc

- ✓The CALL instruction and the subroutine it calls are in the same segment.
- ✓Save the current value of the IP on the stack.
- ✓load the subroutine's offset into IP (nextinst + offset)

Calling Program	Subroutine	Stack

Main proc	sub1 proc
001A: call sub1	0080: mov ax,1
001D: inc ax	…
.	ret
Main endp	sub1 endp

1ffd	1D
1ffe	00
1fff	(not used)

Calling a FAR proc

- ✓The CALL instruction and the subroutine it calls are in the "Different" segments.
- ✓Save the current value of the CS and IP on the stack.
- ✓Then load the subroutine's CS and offset into IP.

Calling Program	Subroutine	Stack

```
          Main proc                          sub1 proc far
1FCB:001A: call far ptr sub1       ->  4EFA:0080: mov ax,1
1FCB:001F: inc ax                      ....
          ...                          ....
          ...                          ret (retf opcode generated)
          Main endp                    sub1 endp
```

1ffb	1F	IP
1ffc	00	IP
1ffd	CB	SEG
1ffe	1F	SEG
1fff	N/A	

Opcode 8000 FA4E

Example on Far/Near Procedure Calls

```
0350:1C00 Call FarProc
0350:1C05 Call NearProc
0350:1C08 nop
```

1ff0	08
1ffa	1C
1ffb	05
1ffc	1C
1ffd	50
1ffe	03
1fff	X

Nested Procedure Calls

A subroutine may itself call other subroutines.

Example:

```
        main proc
000A    call subr1
000C    mov ax,…
…

        main endp
```

```
        subr2  proc
0050    nop
        …
        call subr3
0060    ret …

        subr2 endp
```

```
        subr1  proc
0030    nop
        …
        call subr2
0040    ret …

        subr1 endp
```

```
        subr3  proc
0070    nop
        …
0079    nop
007A    ret
        subr3 endp
```

Q: show the stack contents at 0079?

1ff0	60
1ffa	00
1ffb	40
1ffc	00
1ffd	0c
1ffe	00
1fff	X

Do NOT overlap Procedure Declarations

Push and Pop Instructions

To save registers and parameters on the stack	PUSH XX PUSH YY PUSH ZZ
Main body of the subroutine	. . . . . .
To restore registers and parameters from the stack	POP ZZ POP YY POP XX
Return to main program	RET

Push S (16/32 bit or Mem)
(SP) ← (SP) - 2
((SP)) ← (S)

Pop D (16/32 bit or Mem)
(D) ← ((SP))
(SP) ← (SP) + 2

Loop and Loop Handling Instructions

Mnemonic	Meaning	Format	Operation
LOOP	Loop	LOOP Short-label	(CX) ← (CX) − 1 Jump is initiated to location defined by short-label if (CX) ≠ 0; otherwise, execute next sequential instruction
LOOPE/LOOPZ	Loop while equal/ loop while zero	LOOPE/LOOPZ Short-label	(CX) ← (CX) − 1 Jump to location defined by short-label if (CX) ≠ 0 and (ZF) = 1; otherwise, execute next sequential instruction
LOOPNE/ LOOPNZ	Loop while not equal/ loop while not zero	LOOPNE/LOOPNZ Short-label	(CX) ← (CX) − 1 Jump to location defined by short-label if (CX) ≠ 0 and (ZF) = 0; otherwise, execute next sequential instruction

Figure 6–28 Loop instructions.

Loop

```
NEXT:   MOV CX,COUNT        Load count for the number of repeats
        .
        .
        .
        .                   Body of routine that is repeated
        .
        .
        .
        LOOP NEXT           Loop back to label NEXT if count not zero
```

(a)

```
        MOV     AX,DATASEGADDR
        MOV     DS,AX
        MOV     SI,BLK1ADDR
        MOV     DI,BLK2ADDR
        MOV     CX,N
NXTPT:  MOV     AH,[SI]
        MOV     [DI],AH
        INC     SI
        INC     DI
        LOOP    NXTPT
        HLT
```

(b)

Nested Loops

single Loop	Nested Loops

```
MOV CX,A
BACK: …
…
…
…
LOOP BACK
```

```
        MOV CX,A
OUTER:  PUSH CX
        MOV CX, 99
INNER:  NOP
        …
        …
        …
        LOOP INNER
        POP CX
        LOOP OUTER
```

How many times will the loop execute, if JCXZ wasn't there

```
MOV CX,0
DLOOP: JCXZ SKIP ;guarding
BACK: MUL AX,2H
ADD AX,05H
LOOP BACK
SKIP: INC AX; if CX=0
```

INT

INT operates similar to Call

- ❖Processor first pushes the flags
- ❖Trace Flag and Interrupt-enable flags are cleared
- ❖Next the processor pushes the current CS register onto the stack
- ❖Next the IP register is pushed

Example: What is the sequence of events for INT 08? If it generates a CS:IP of 0100:0200. The flag is 0081H.

SP-6	00
SP-5	02
SP-4	00
SP-3	01
SP-2	81
SP-1	00

SP initial

MEMORY / ISR table		
00020	10	IP
00021	00	
00022	80	SEG
00023	05	

0580:
0010

IRET

- IRET must be used for special handling of the stack.
- Must be used at the end of an ISR

SP-6	00
SP-5	02
SP-4	00
SP-3	01
SP-2	81
SP-1	00

SP initial

Return address + flags are loaded

String Instructions

80x86 is equipped with special instructions to handle string operations

String: A series of data words (or bytes) that reside in consecutive memory locations

Operations: move, scan, compare

String Instruction:

Byte transfer, SI or DI increment or decrement by 1

Word transfer, SI or DI increment or decrement by 2

DWord transfer, SI or DI increment or decrement by 4

String Instructions - D Flag

The Direction Flag: Selects the auto increment D=0 or the auto decrement D=1 operation for the DI and SI registers during string operations. D is used only with strings

Mnemonic	Meaning	Format	Operation	Flags Affected
CLD	Clear DF	CLD	(DF) ← 0	DF
STD	Set DF	STD	(DF) ← 1	DF

CLD → Clears the D flag / STD → Sets the D flag

String Instructions

Mnemonic	Meaning	Format	Operation	Flags Affected
MOVS	Move string	MOVSB/MOVSW	((ES)0 + (DI)) ← ((DS)0 + (SI)) (SI) ← (SI) ± 1 or 2 (DI) ← (DI) ± 1 or 2	None
CMPS	Compare string	CMPSB/CMPSW	Set flags as per ((DS)0 + (SI)) − ((ES)0 + (DI)) (SI) ← (SI) ± 1 or 2 (DI) ← (DI) ± 1 or 2	CF, PF, AF, ZF, SF, OF
SCAS	Scan string	SCASB/SCASW	Set flags as per (AL or AX) − ((ES)0 + (DI)) (DI) ← (DI) ± 1 or 2	CF, PF, AF, ZF, SF, OF
LODS	Load string	LODSB/LODSW	(AL or AX) ← ((DS)0 + (SI)) (SI) ← (SI) ± 1 or 2	None
STOS	Store string	STOSB/STOSW	((ES)0 + (DI)) ← (AL or AX) ± 1 or 2 (DI) ← (DI) ± 1 or 2	None

```
        MOV     AX,DATASEGADDR
        MOV     DS,AX
        MOV     ES,AX
        MOV     SI,BLK1ADDR
        MOV     DI,BLK2ADDR
        MOV     CX,N
        CLD
NXTPT:  MOVSB
        LOOP    NXTPT
        HLT
```

Repeat String REP

Basic string operations must be repeated in order to process arrays of data; this is done by inserting a repeat prefix.

Prefix	Used with:	Meaning
REP	MOVS STOS	Repeat while not end of string $CX \neq 0$
REPE/REPZ	CMPS SCAS	Repeat while not end of string and strings are equal $CX \neq 0$ and $ZF = 1$
REPNE/REPNZ	CMPS SCAS	Repeat while not end of string and strings are not equal $CX \neq 0$ and $ZF = 0$

Figure 6–36 Prefixes for use with the basic string operations.

Example. Find and replace

- Write a program that scans the name “Mr.Gohns” and replaces the “G” with the letter “J”.

search.asm

```
Data1 db  'Mr.Gones','$‘
.code
mov es,ds
cld ;set auto increment bit D=0
mov di, offset data1
mov cx,09; number of chars to be scanned
mov al,'G'; char to be compared against
repne SCASB; start scan AL =? ES[DI]
jne Over; if Z=0
dec di; Z=1
mov byte ptr[di], 'J'
Over:  mov ah,09
mov dx,offset data1
int 21h; display the resulting String
```

Search.exe

Strings into Video Buffer

Fill the Video Screen with a value

Clear.exe

```
CLD
MOV AX,0B800H
MOV ES,AX
MOV DI,0
MOV CX,2000H
MOV AL,20h
REP STOSW
```

Example. Display the ROM BIOS Date

- Write an 8086 program that searches the BIOS ROM for its creation date and displays that date on the monitor.
- If a date cannot be found display the message “date not found”
- Typically the BIOS ROM date is stored in the form xx/xx/xx beginning at system address F000:FFF5
- Each character is in ASCII form and the entire string is terminated with the null character (00)
- Add a ‘$’ character to the end of the string and make it ready for DOS function 09, INT 21

Date.asm