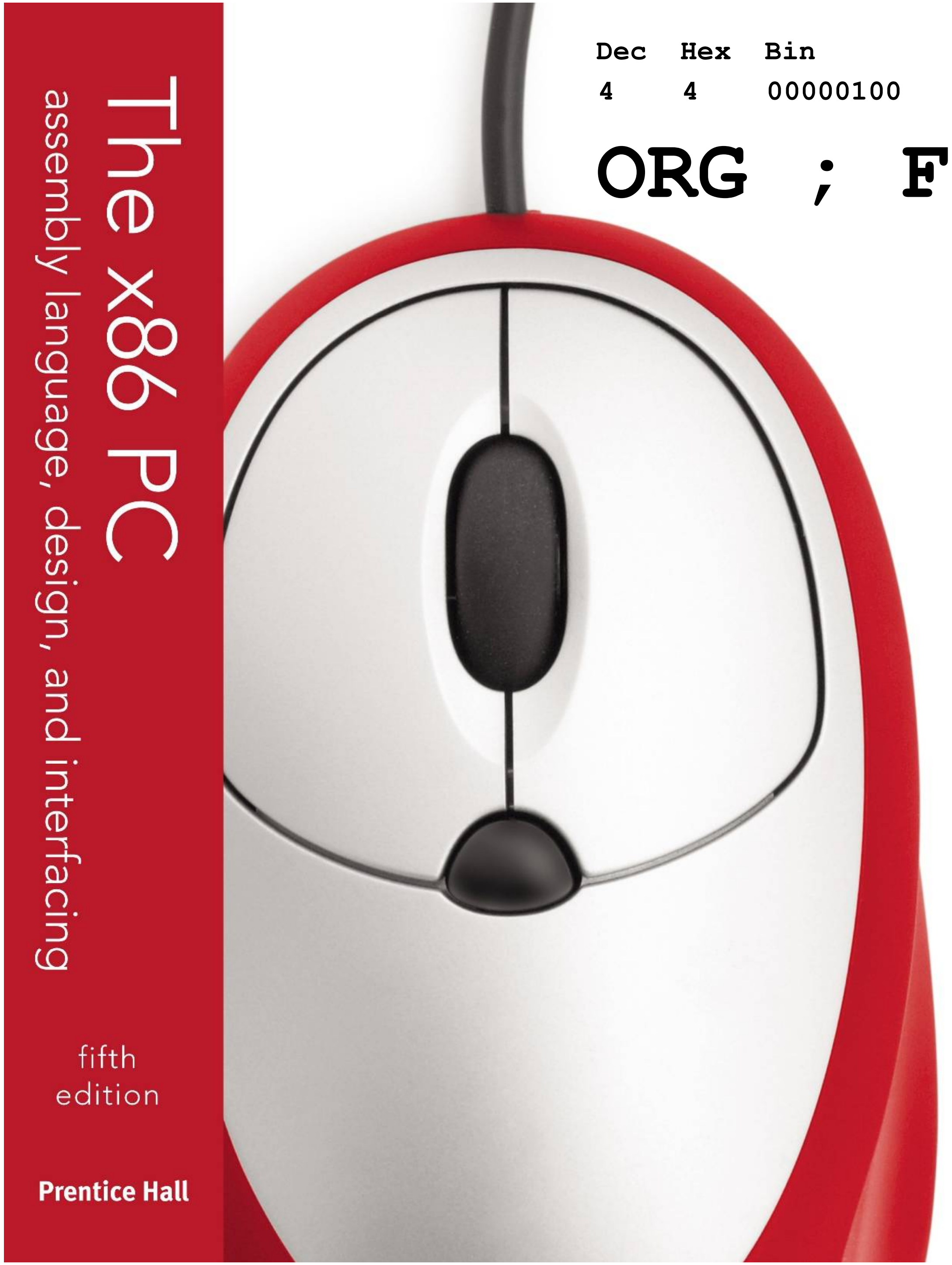
Dec	Hex	Bin
4	4	00000100

ORG ; FIVE

OTHER COMMANDS
Conditions
Strings
etc.



The x86 PC
assembly language, design, and interfacing
fifth edition

MUHAMMAD ALI MAZIDI
JANICE GILLISPIE MAZIDI
DANNY CAUSEY

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'

Data Transfer Instructions - XCHG

Mnemonic	Meaning	Format	Operation	Flags Affected
XCHG	Exchange	XCHG D,S	(Dest) ↔ (Source)	None

Destination	Source
Reg16	Reg16
Memory	Register
Register	Register
Register	Memory

Example: XCHG [1234h], BX

Data Transfer Instructions – LEA, LDS, LES

Mne monic	Meaning	Format	Operation	Flags Affected
LEA	Load Effective Address	LEA Reg16,EA	EA →(Reg16)	None
LDS	Load Register and DS	LDS Reg16, MEM32	(Mem32) → (Reg16) (Mem32 + 2) → (DS)	None
LES	Load Register and ES	LES Reg16, MEM32	(Mem32) → (Reg16) (Mem32 + 2) → (ES)	None

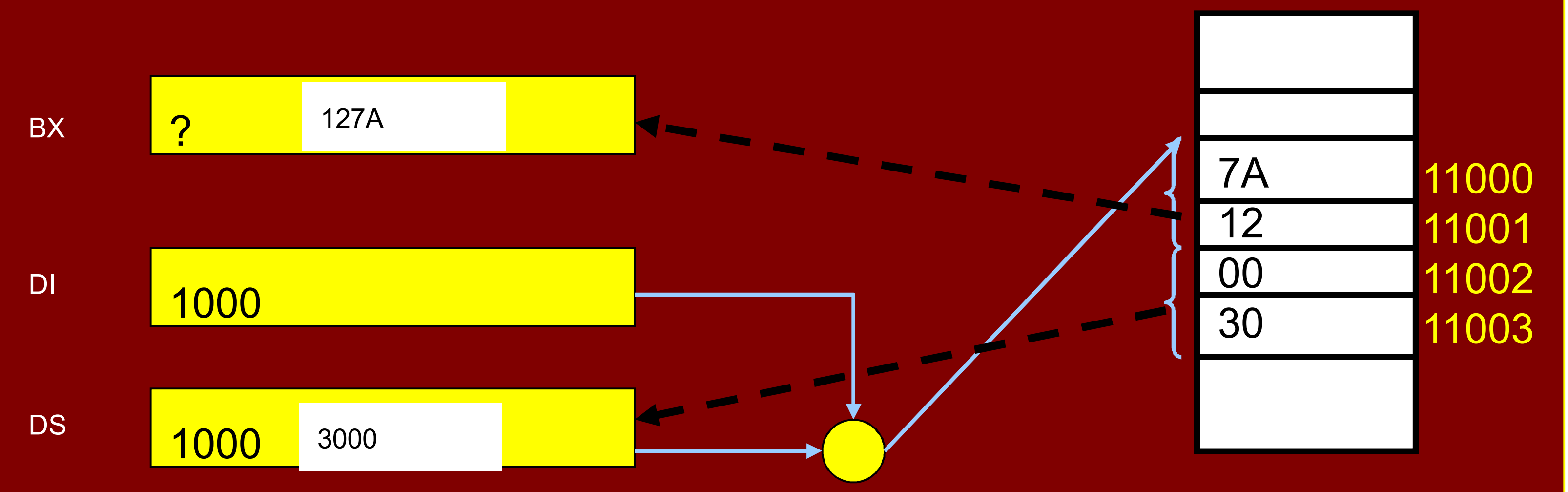
Examples for LEA, LDS, LES

```
DATAX  DW 1000H
DATAY  DW 5000H
.CODE
LEA SI, DATAX
MOV DI, OFFSET DATAY; THIS IS MORE EFFICIENT

LEA BX,[DI]; IS THE SAME AS…
MOV BX,DI; THIS JUST TAKES LESS CYCLES.

LEA BX,DI; INVALID!
```

LDS BX, [DI];



Flag Control Instructions

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

Bulk manipulation of the flags

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

Individual manipulation of the flags

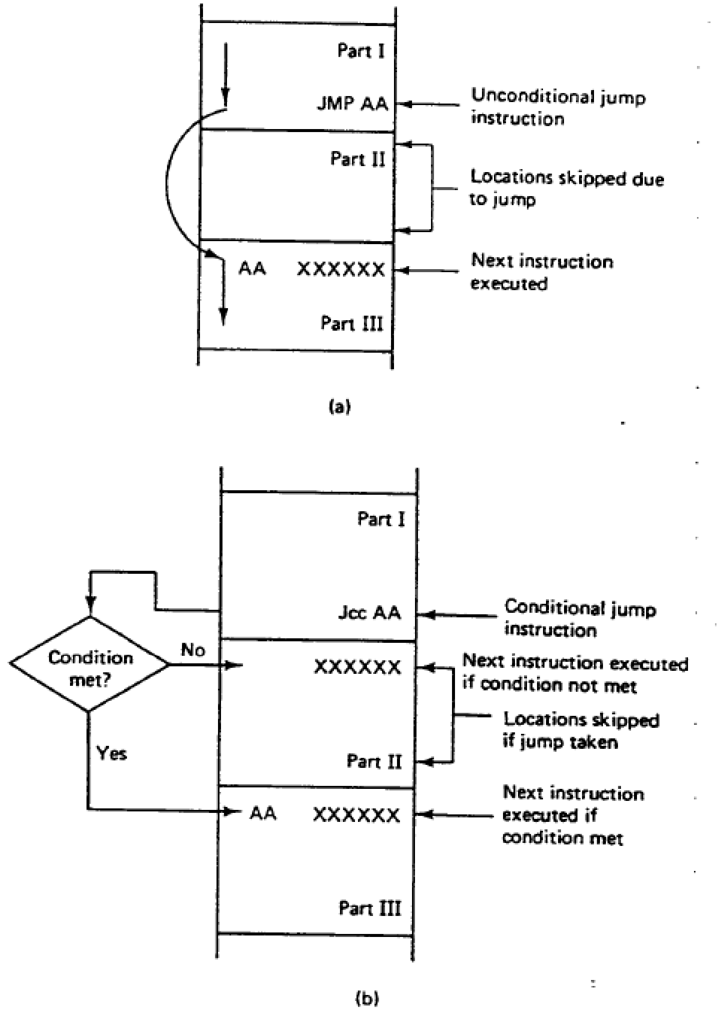
- Example (try with debug)

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

  - Check the flag changes!

Jump Instructions

- Unconditional vs conditional jump



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

SJ Example

Hello2.exe


```
.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
```

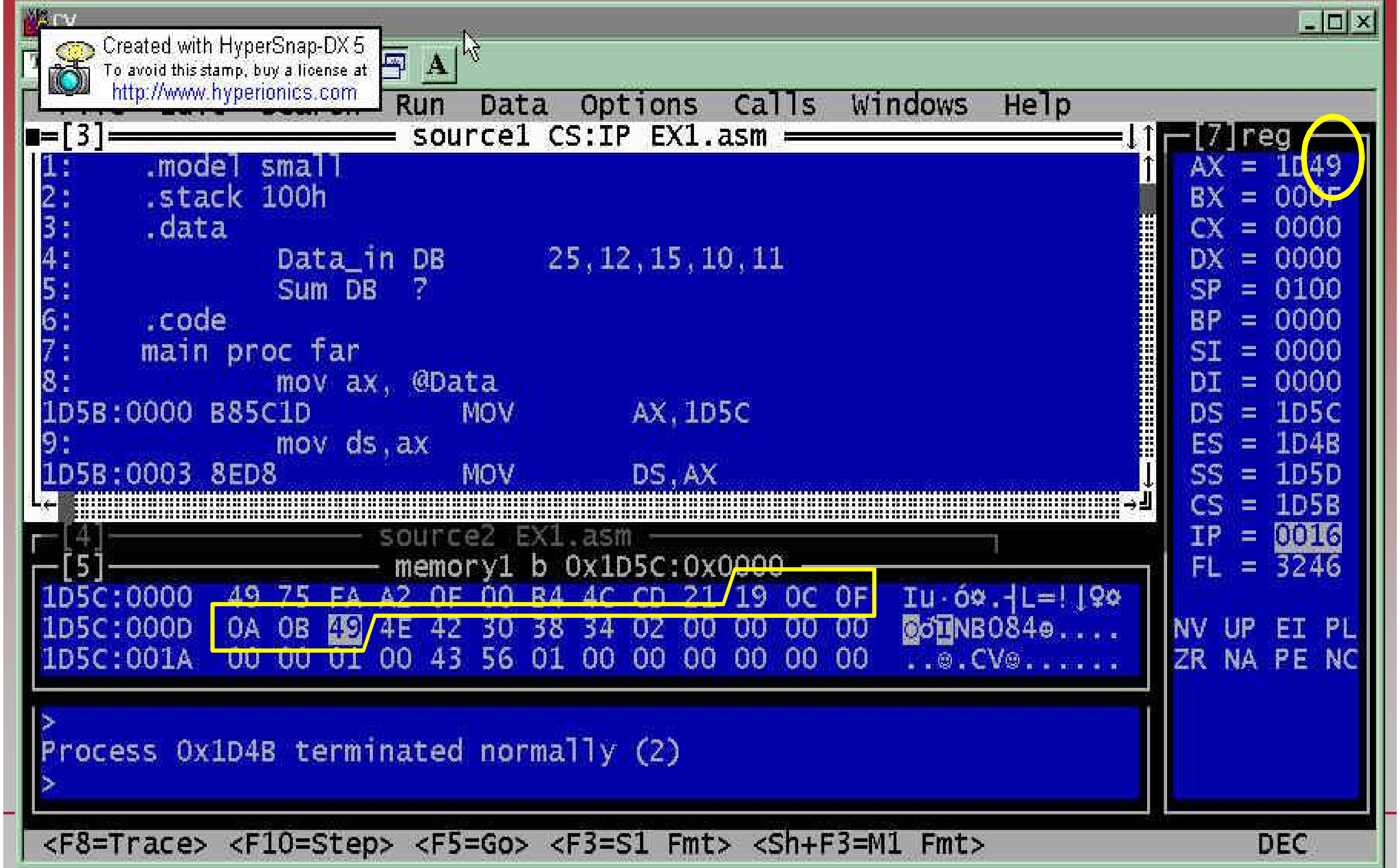
A Simple Example Program finds the sum

- 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

Nested Loops

single Loop

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

Nested Loops

```
            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
```

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

```
        MOV CX,COUNT      Load count for the number of repeats
NEXT:   .
        .                 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)

3.4: BCD AND ASCII CONVERSION
BCD number system

- BCD stands for binary coded decimal.
  - Needed because we use the digits 0 to 9 for numbers in everyday life.
  - Computer literature features two terms for BCD numbers:
    - Unpacked BCD.
    - Packed BCD.

Digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

3.4: BCD AND ASCII CONVERSION
BCD unpacked vs. packed

- In unpacked BCD, the lower 4 bits of the number represent the BCD number.
  - The rest of the bits are 0.
    - "0000 1001" and "0000 0101" are unpacked BCD for 9 & 5.
  - Unpacked BCD it takes 1 byte of memory location.
    - Or a register of 8 bits to contain the number.
- In packed BCD, a single byte has two BCD numbers.
  - One in the lower 4 bits; One in the upper 4 bits.
    - "0101 1001" is packed BCD for 59.
  - As it takes only 1 byte of memory to store the packed BCD operands, it is twice as efficient in storing data.

3.4: BCD AND ASCII CONVERSION
ASCII numbers

- In ASCII keyboards, when key "0" is activated "**011 0000**" (30H) is provided to the computer.
  - 31H (011 0001) is provided for key "1", etc.

Key	ASCII (hex)	Binary	BCD (unpacked)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

- To convert ASCII data to BCD, removed the tagged "011" in the higher 4 bits of the ASCII.
  - Each ASCII number is ANDed with "0000 1111". (0FH)

3.4: BCD AND ASCII CONVERSION
ASCII to unpacked BCD conversion

- Programs **3-5a**, 3-5b, and 3-5c show three methods for converting the 10 ASCII digits to unpacked BCD.
  - Using this data segment:

```
ASC         DB      '9562481273'
            ORG     0010H
UNPACK      DB      10 DUP(?)
```

The data is defined as DB, a byte definition directive, and is accessed in word-sized chunks.

```
            MOV     CX,5
            MOV     BX,OFFSET ASC       ;BX points to ASCII data
            MOV     DI,OFFSET UNPACK    ;DI points to unpacked BCD data
AGAIN:      MOV     AX,[BX]             ;move next 2 ASCII numbers to AX
            AND     AX,0F0FH            ;remove ASCII 3s
            MOV     [DI],AX             ;store unpacked BCD
            ADD     DI,2                ;point to next unpacked BCD data
            ADD     BX,2                ;point to next ASCII data
            LOOP    AGAIN
```

Program 3-5a

3.4: BCD AND ASCII CONVERSION
ASCII to unpacked BCD conversion

- Programs 3-5a, **3-5b**, and 3-5c show three methods for converting the 10 ASCII digits to unpacked BCD.
  - Using this data segment:

```
ASC       DB    '9562481273'
          ORG   0010H
UNPACK    DB    10 DUP(?)
```

Using the PTR directive as shown, makes the code more readable for programmers.

```
        MOV   CX,5               ;CX is loop counter
        MOV   BX,OFFSET ASC      ;BX points to ASCII data
        MOV   DI,OFFSET UNPACK   ;DI points to unpacked BCD data
AGAIN:  MOV   AX,WORD PTR [BX]   ;move next 2 ASCII numbers to AX
        AND   AX,0F0FH           ;remove ASCII 3s
        MOV   WORD PTR [DI],AX   ;store unpacked BCD
        ADD   DI,2               ;point to next unpacked BCD data
        ADD   BX,2               ;point to next ASCII data
        LOOP  AGAIN
```

Program 3-5b

3.4: BCD AND ASCII CONVERSION
ASCII to unpacked BCD conversion

- Programs 3-5a, 3-5b, and **3-5c** show three methods for converting the 10 ASCII digits to unpacked BCD.
  - Using this data segment:

```
ASC         DB      '9562481273'
            ORG     0010H
UNPACK      DB      10 DUP(?)
```

3-5c uses based addressing mode since BX+ASC is used as a pointer.

```
        MOV   CX,10             ;load the counter
        SUB   BX,BX             ;clear BX
AGAIN:  MOV   AL,ASC[BX]        ;move to AL content of mem [BX+ASC]
        AND   AL,0FH            ;mask the upper nibble
        MOV   UNPACK[BX],AL     ;move to mem [BX+UNPACK] the AL
        INC   BX                ;point to next byte
        LOOP  AGAIN             ;loop until it is finished
```

Program 3-5c

3.4: BCD AND ASCII CONVERSION
ASCII/BCD conversions

- To convert ASCII to packed BCD, it is converted to unpacked BCD (eliminating the 3), then combined to make packed BCD.
- To convert packed BCD to ASCII, it must first be converted to unpacked.
  - The unpacked BCD is tagged with 011 0000 (30H).

3.4: BCD AND ASCII CONVERSION
ASCII to packed BCD conversion

- For 4 & 7, the keyboard gives 34 & 37, respectively.
  - The goal is to produce packed BCD 47H or “0100 0111“.

Key	ASCII	Unpacked BCD	Packed BCD
4	34	00000100	
7	37	00000111	01000111 or 47H

```
                ORG     0010H
VAL_ASC         DB      '47'
VAL_BCD         DB      ?
;reminder:      DB will put 34 in 0010H location and 37 in 0011H
                MOV     AX,WORD PTR VAL_ASC     ;AH=37,AL=34
                AND     AX,0F0FH            ;mask 3 to get unpacked BCD
                XCHG    AH,AL               ;swap AH and AL.
                MOV     CL,4                ;CL=04 to shift 4 times
                SHL     AH,CL               ;shift left AH to get AH=40H
                OR      AL,AH               ;OR them to get packed BCD
                MOV     VAL_BCD,AL          ;save the result
```

3.4: BCD AND ASCII CONVERSION
packed BCD to ASCII conversion

- Converting from packed BCD to ASCII.

Packed BCD	Unpacked BCD	ASCII
29H	02H & 09H	32H & 39H
0010 1001	0000 0010 & 0000 1001	011 0010 & 011 1001

```
VAL1_BCD    DB     29H
VAL3-ASC    DW     ?
            ...
            MOV    AL,VAL1_BCD
            MOV    AH,AL        ;copy AL to AH. now AH=29,AL=29H
            AND    AX,0F00FH    ;mask 9 from AH and 2 from AL
            MOV    CL,4         ;CL=04 for shift
            SHR    AH,CL        ;shift right AH to get unpacked BCD
            OR     AX,3030H     ;combine with 30 to get ASCII
            XCHG   AH,AL        ;swap for ASCII storage convention
            MOV    VAL3_ASC,AX  ;store the ASCII
```

AAA

Ex. ASCII CODE 0-9 = 30h –> 39h

MOV AX, 38H ;(ASCII code for number 8)

ADD AL, 39H ;(ASCII code for number 9)

AAA; used for addition AX has → 0107

ADD AX, 3030H; change answer to ASCII if you needed

3.4: BCD AND ASCII CONVERSION
BCD addition and subtraction

- After adding packed BCD numbers, the result is no longer BCD.

```
MOV AL,17H
ADD AL,28H
```

← Adding them gives 0011 1111B (3FH). (not BCD)

  - The result should have been 17 + 28 = 45 (0100 0101).
    - To correct, add 6 (0110) to the low digit: 3F + 06 = 45H.
  - The same could have happened in the upper digit.
    - This problem is so pervasive that the vast majority of microprocessors have an instruction to deal with it.

3.4: BCD AND ASCII CONVERSION
DAA

- DAA (decimal adjust for addition) is provided in the x86 for correcting the BCD addition problem.
  - DAA will add 6 to the lower, or higher nibble if needed
    - Otherwise, it will leave the result alone.

```
DATA1    DB   47H
DATA2    DB   25H
DATA3    DB?
         MOV  AL,DATA1       ;AL holds first BCD operand
         MOV  BL,DATA2       ;BL holds second BCD operand
         ADD  AL,BL          ;BCD addition
         DAA                 ;adjust for BCD addition
         MOV  DATA3,AL       ;store result in correct BCD form
```

After execution, DATA3 will contain 72H.

3.4: BCD AND ASCII CONVERSION
DAA general rules & summary

- General rules for DAA:
  - The source can be an operand of any addressing mode.
    - The destination must be AL in order for DAA to work.
  - DAA must be used after the addition of BCD operands.
    - BCD operands can never have any digit greater than 9.
  - DAA works only after an ADD instruction.
    - It will not work after the INC instruction.
- After an ADD or ADC instruction:
  - If the lower nibble (4 bits) is greater than 9, or if AF = 1.
    - Add 0110 to the lower 4 bits.
  - If the upper nibble is greater than 9, or if CF = 1.
    - Add 0110 to the upper nibble.

3.4: BCD AND ASCII CONVERSION
DAA summary of action

Use of DAA after adding multibyte packed BCD numbers.

Two sets of ASCII data have come in from the keyboard. Write and run a program to:
1. Convert from ASCII to packed BCD.
2. Add the multibyte packed BCD and save it.
3. Convert the packed BCD result to ASCII.

```
TITLE       PROG3-6 (EXE) ASCII TO BCD CONVERSION AND ADDITION
PAGE        60,132
.MODE  SMALL
.STACK 64
;---------------------
            .DATA
DATA1_ASC   DB     `0649147816'
            ORG    0010H
DATA2_ASC   DB     `0072687188'
            ORG    0020H
DATA3_BCD   DB     5 DUP (?)
            ORG    0028H
DATA4_BCD   DB     5 DUP (?)
            ORG    0030H
```

Program 3-6

See the entire program listing on pages 116-117 of your textbook.

DAA Example

Ex. 4 AL contains 25 (packed BCD)
BL contains 56 (packed BCD)

ADD AL, BL
DAA

25
56
\+ ----------
7B → 81

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
```

```
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
```

6	?	6	4	+	8	9	=

3.4: BCD AND ASCII CONVERSION
BCD subtraction and correction

- DAS (decimal adjust for subraction) is provided in the x86 for correcting the BCD subtraction problem.
  - When subtracting packed BCD (single-byte or multibyte) operands, the DAS instruction is used after SUB or SBB.
    - AL must be used as the destination register.
- After a SUB or SBB instruction:
  - If the lower nibble is greater than 9, or if AF = 1.
    - Subtract 0110 from the lower 4 bits.
  - If the upper nibble is greater than 9, or CF = 1.
    - Subtract 0110 from the upper nibble.

BCD subtraction and correction

- Due to the widespread use of BCD numbers, a specific data directive, DT, has been created.
  - To represent BCD numbers 0 to 1020 - 1. (twenty 9s)

```
BUDGET       DT    87965141012
EXPENSES     DT    31610640392
BALANCE      DT    ?                ;balance = budget - expenses

      MOV    CX,10             ;counter=10
      MOV    BX,00             ;pointer=0
      CLC                      ;clear carry for the 1st iteration
      BACK:  MOVAL,BYTE PTR BUDGET[BX] ;get a byte of the BUDGET
      SBB    AL,BYTE PTR EXPENSES[BX]  ;subtract a byte from it
      DAS                              ;correct result for BCD
      MOV    BYTE PTR BALANCE[BX],AL ;save it in BALANCE
      INC    BX                ;increment for the next byte
      LOOP   BACK              ;continue until CX=0
```

BCD and ASCII Numbers

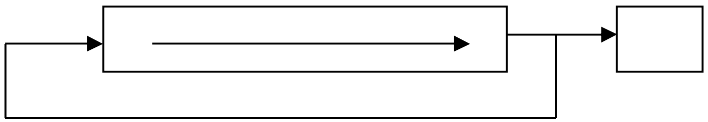
- BCD (Binary Coded Decimal)
  - Unpacked BCD: One byte per digit
  - Packed BCD: 4 bits per digit (more efficient in storing data)
- ASCII to unpacked BCD conversion
  - Keyboards, printers, and monitors all use ASCII.
  - Digits 0 to 9 are represented by ASCII codes 30 – 39.

- 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
```



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