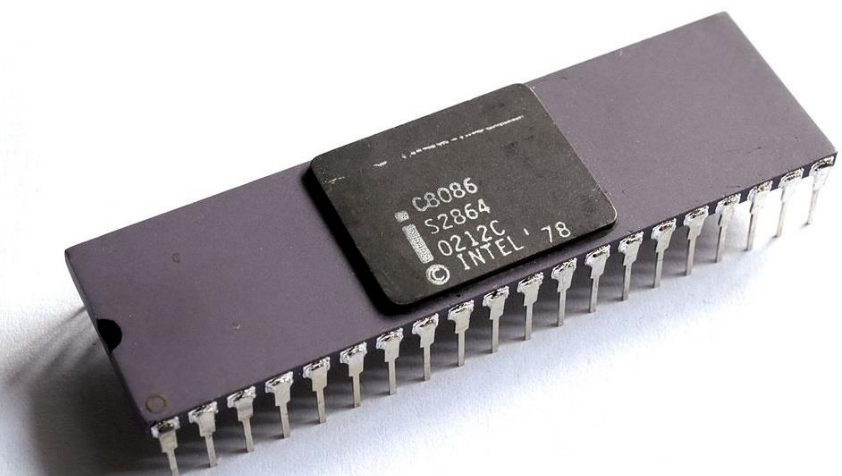
ساختار و زبان کامپیوتر

فصل هشتم

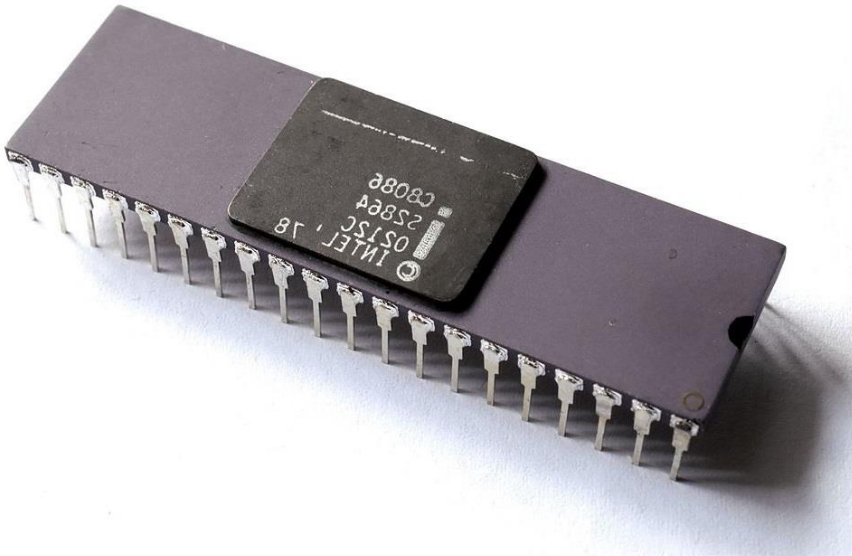
معماری و اسمبلی ۸۰۸۶



Computer Structure and Machine Language

Chapter Eight

8086 Instruction Set Architecture



Copyright Notice

Parts (text & figures) of this lecture are adopted from:

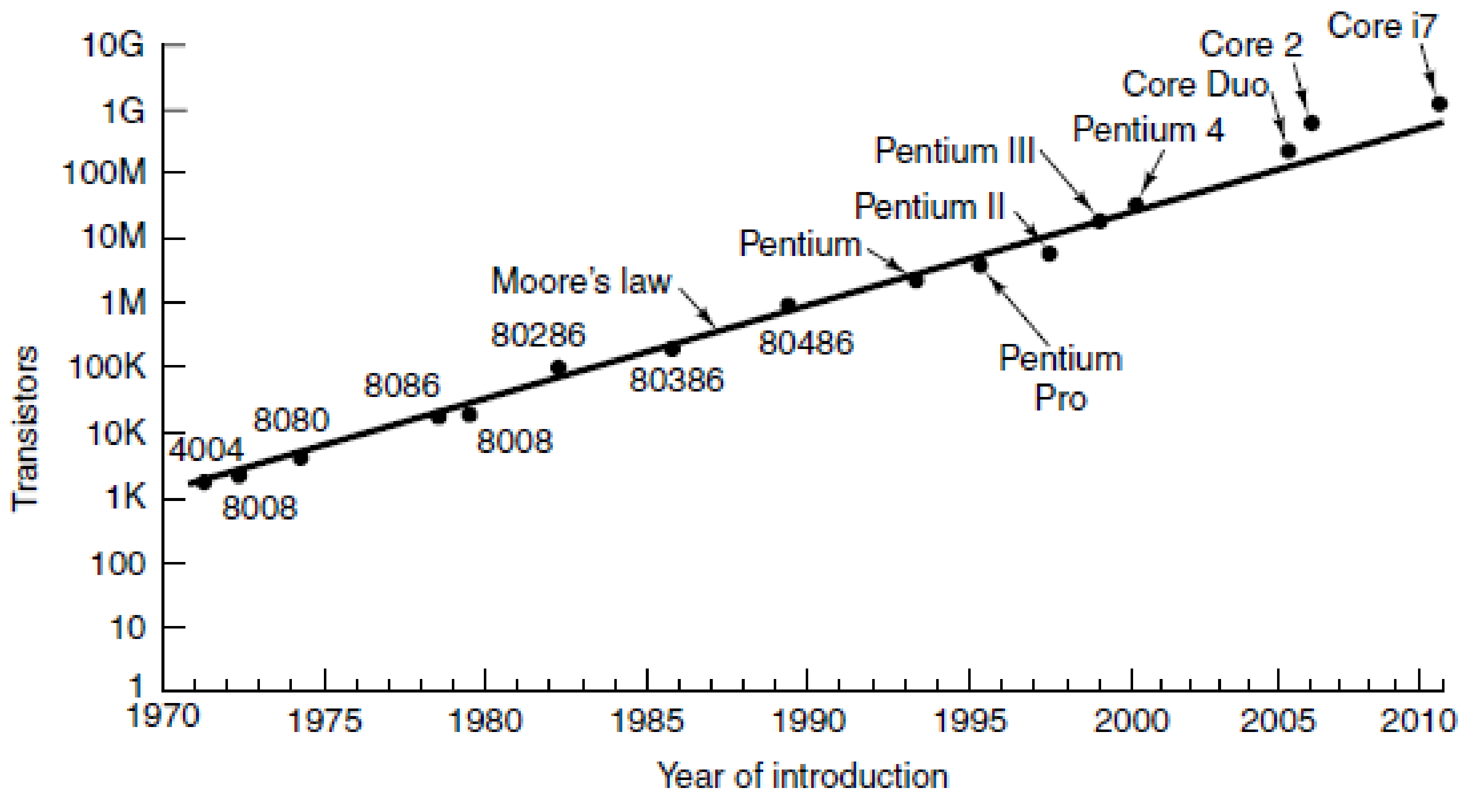
- "The 80x86 IBM PC and Compatible Computers, Vol. I & II", 4th Ed., M. Mazidi & J. Mazidi, Pearson, 2003
- M. Rafiquzzaman, "Microprocessors and Microcomputer-Based System Design", 2nd Ed., CRC Press, 1995
- D. Patterson & J. Hennessey, "Computer Organization & Design, The Hardware/Software Interface", 6th Ed., MK publishing, 2020

Contents

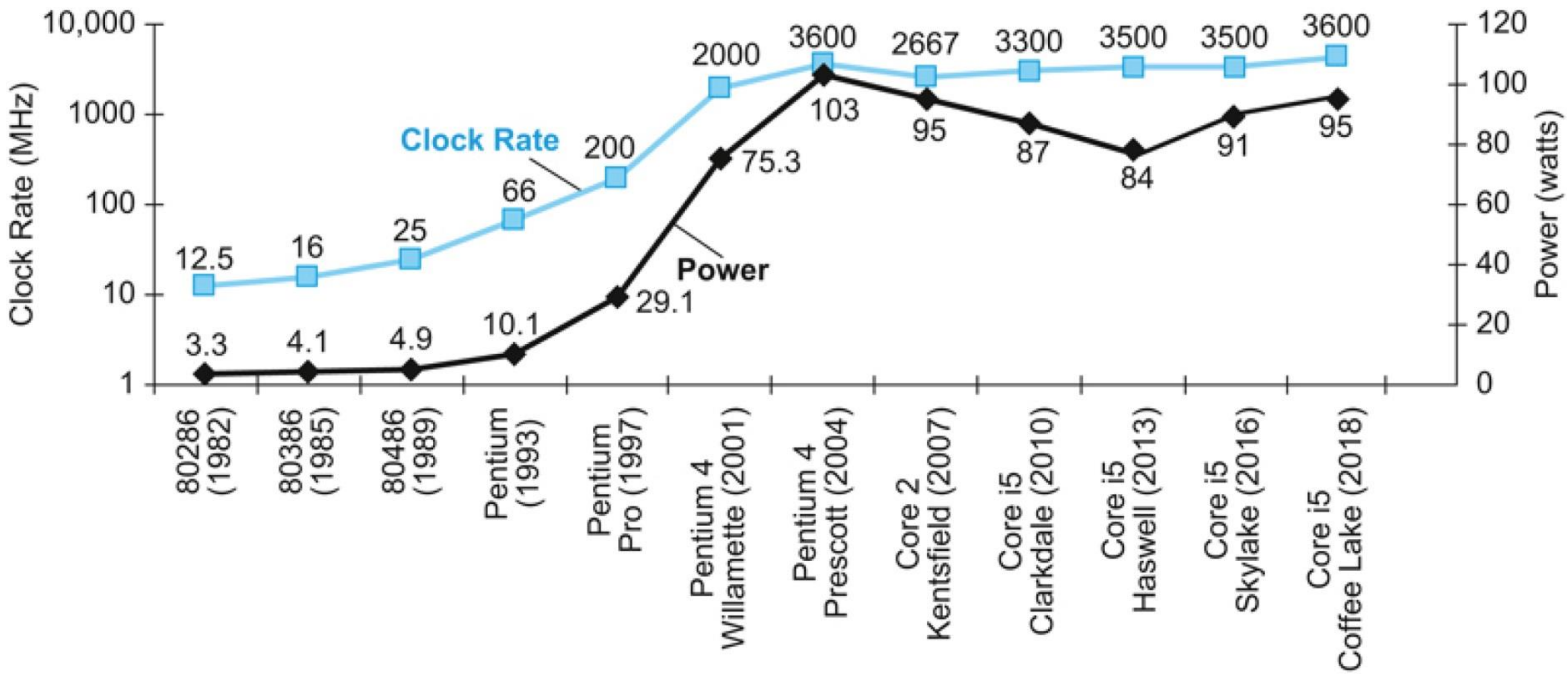
The Intel Family History

Chip	Date	MHz	Trans.	Memory	Notes
4004	4/1971	0.108	2300	640	First microprocessor on a chip
8008	4/1972	0.108	3500	16 KB	First 8-bit microprocessor
8080	4/1974	2	6000	64 KB	First general-purpose CPU on a chip
8086	6/1978	5–10	29,000	1 MB	First 16-bit CPU on a chip
8088	6/1979	5–8	29,000	1 MB	Used in IBM PC
80286	2/1982	8–12	134,000	16 MB	Memory protection present
80386	10/1985	16–33	275,000	4 GB	First 32-bit CPU
80486	4/1989	25–100	1.2M	4 GB	Built-in 8-KB cache memory
Pentium	3/1993	60–233	3.1M	4 GB	Two pipelines; later models had MMX
Pentium Pro	3/1995	150–200	5.5M	4 GB	Two levels of cache built in
Pentium II	5/1997	233–450	7.5M	4 GB	Pentium Pro plus MMX instructions
Pentium III	2/1999	650–1400	9.5M	4 GB	SSE Instructions for 3D graphics
Pentium 4	11/2000	1300–3800	42M	4 GB	Hyperthreading; more SSE instructions
Core Duo	1/2006	1600–3200	152M	2 GB	Dual cores on a single die
Core	7/2006	1200–3200	410M	64 GB	64-bit quad core architecture
Core i7	1/2011	1100–3300	1160M	24 GB	Integrated graphics processor

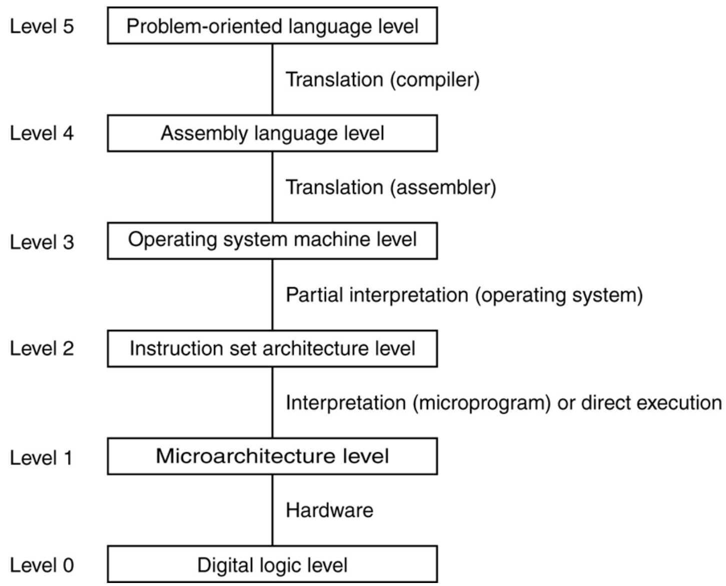
Moore's Law for Intel CPU Chips

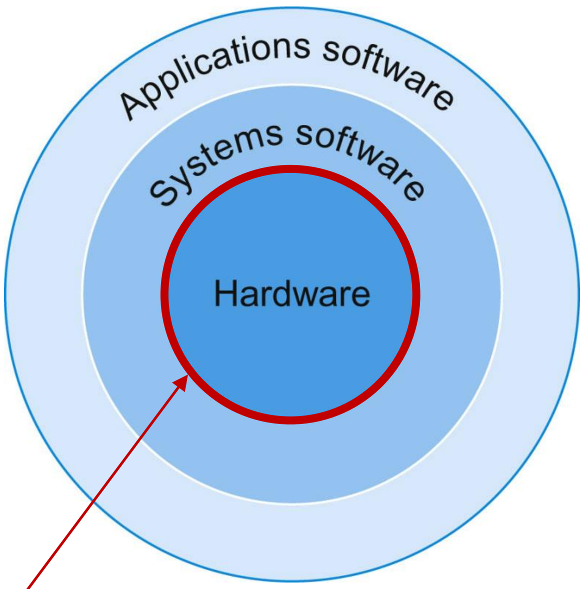


Clock rate & Power for ...



Hierarchical Levels (Reminder)





Tanenbaum, Structured Computer Organization, 5th Edition

Instruction Set Architecture (ISA)

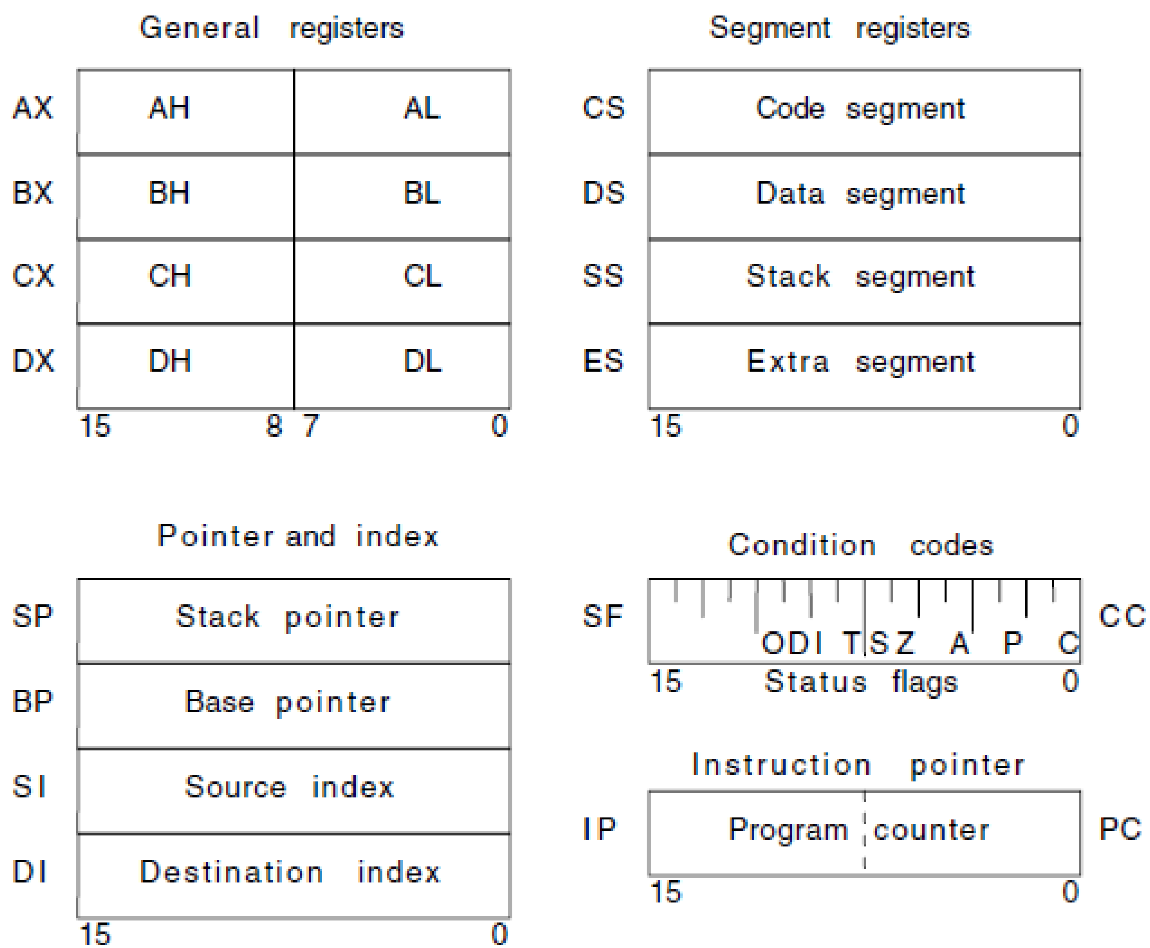
Instruction Set Architecture (ISA)

- How the machine appears to a machine language programmer
- Specifies:
  - Registers
  - Memory Models
  - Addressing Modes
  - Available instructions

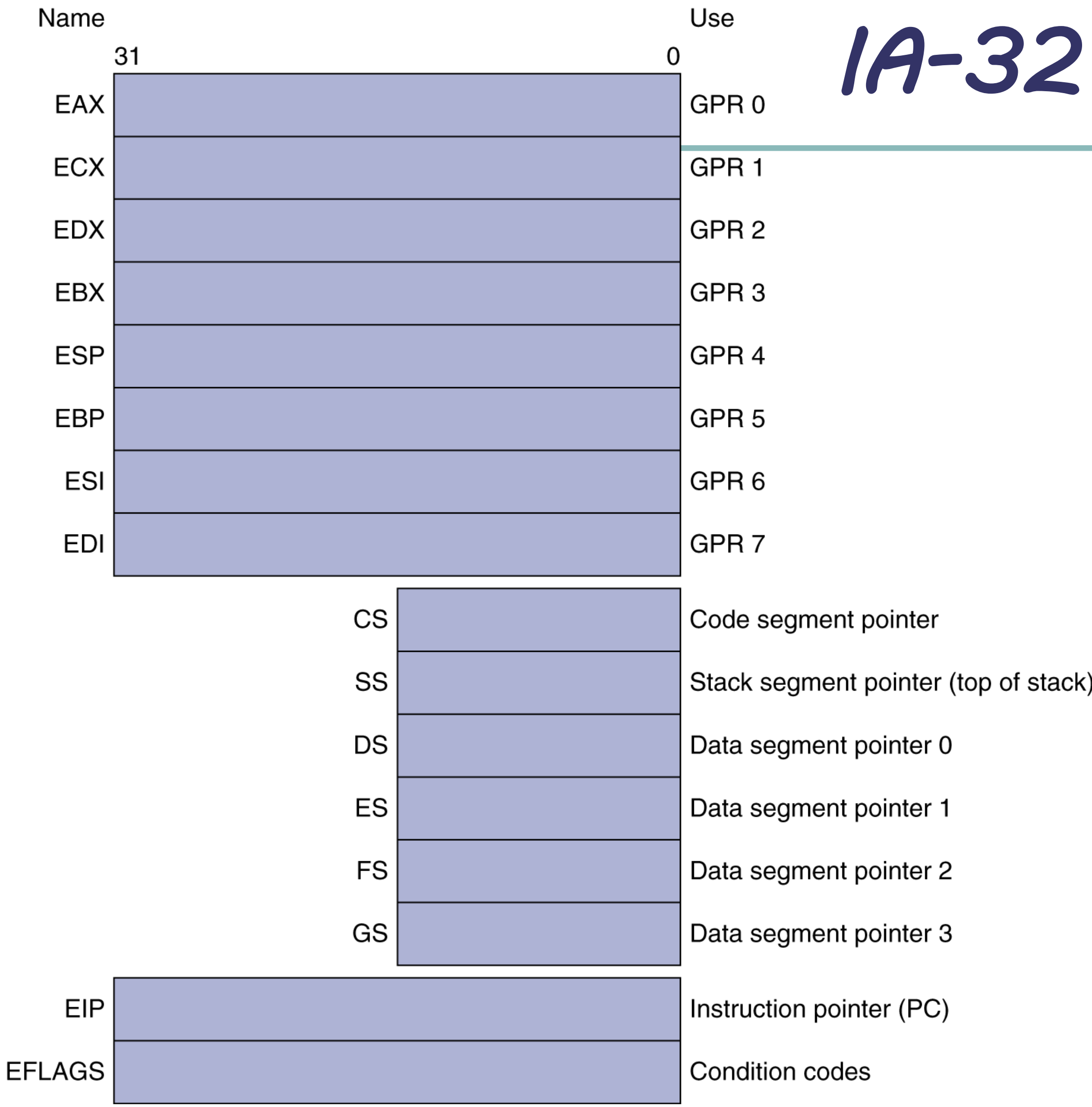
Fundamentals

- Registers
- Basic Instructions
- An Assembly Program
- Directives
- Examples
- Interrupt 21H

8086/88 Registers



IA-32 Registers



The MOV Instruction

- **MOV dest, source**
  - *reg, reg*
  - *mem, reg*
  - *reg, mem*
  - *mem, immed*
  - *reg, immed*
- Sizes of both operands must be same

GENERAL REGISTERS	AH	AL
	BH	BL
	CH	CL
	DH	DL
	SP	
	BP	
	SI	
	DI	

Basic Arithmetic Instructions

- **ADD** ***dest, source***
- **SUB** ***dest, source***
- **INC** ***dest***
- **DEC** ***dest***
- **NEG** ***dest***

An Assembly Program Shell

```
; Program description

StSeg    Segment STACK 'STACK'
         DB 100H DUP (?)
StSeg    ENDS

DtSeg    Segment
         ; place data here
DtSeg    ENDS

CDSeg    Segment
         ASSUME CS:CDSeg,DS:DtSeg,SS:StSeg
Start:
         MOV AX,DtSeg     ; set DS to point to the data segment
         MOV DS,AX

         ; type your code here

         MOV AH,4CH   ; DOS: terminate program
         MOV AL,0     ; return code will be 0
         INT 21H      ; terminate the program
CDSeg    ENDS
END Start
```

Defining A Segment

```
label     SEGMENT     [options]
          ;place the statements belonging to this segment here
label     ENDS
```

Example

```
STSEG     SEGMENT          ;the "SEGMENT" directive begins the segment
          DB 64 DUP (?)    ;this segment contains only one line
STSEG     ENDS                  ;the "ENDS" segment ends the segment

DTSEG     SEGMENT
DATA1     DB  52H
DATA2     DB  29H
SUM       DB  ?
DTSEG     ENDS
```

A Sample Line

label opcode operand(s) comment

```
L1:  MOV AX,10  ; move 10 to AX
```

label directive value comment

```
Data1  DB 52H  ; define a byte
```

8086 Directives

- ASSUME
- ORG
- DB (Define Byte)
- DUP (Duplicate)
- DW (Define Word)
- DD (Define Double Words)
- DQ (Define Four Words)
- DT (Define Ten Bytes)
- EQU
- EVEN

```
        ORG     100
aByte   DB      12
aStr    DB      "Salam"
aVec    DB      1,2,3
        EVEN
aSpace  DB      6 DUP(?)
aWord   DW      1A2FH
Cnst    EQU     01011110B
aDD     DD      100000
aDQ     DQ      ?
BCDno   DT      14567
DECno   DT      14567D
```

Add Two Numbers

```
StSeg    Segment STACK 'STACK'
         DB 100H DUP (?)
StSeg    ENDS

DtSeg    Segment
num1     DB 100
num2     DB 27
sum      DB ?
DtSeg    ENDS

CDSeg    Segment
         ASSUME CS:CDSeg,DS:DtSeg,SS:StSeg
Start:
         MOV AX,DtSeg     ; set DS to point to the data segment
         MOV DS,AX

         MOV AL,num1
         ADD AL,num2
         MOV sum,AL

         MOV AH,4CH   ; DOS: terminate program
         MOV AL,0     ; return code will be 0
         INT 21H      ; terminate the program
CDSeg    ENDS
END Start
```

Add Five Consecutive Numbers

```
.MODEL SMALL
.STACK 100H

.DATA
dataIN   DB  1,2,3,1,1
         EVEN
sum      DB  ?

.CODE
start:
         MOV AX,@DATA     ; set DS to point to the data segment
         MOV DS,AX

         MOV CX,5                  ; setup loop counter
         MOV BX, OFFSET dataIN     ; setup data pointer
         MOV AL,0                  ; initilaize AL
AGAIN:   ADD AL,[BX]
         INC BX                    ; make BX point to next data item
         DEC CX                    ; decrement loop counter
         JNZ AGAIN
         MOV sum,AL                ; load result into sum

         MOV AH,4CH   ;DOS: terminate program
         MOV AL,0     ;return code will be 0
         INT 21H      ;terminate the program
END start
```

Add Four Consecutive Numbers

```
.MODEL SMALL

.STACK 100H

.DATA
dataIN    DW   15,185,125,25
sum       DW   ?

.CODE
start:
          MOV AX,@DATA     ; set DS to point to the data segment
          MOV DS,AX

          MOV CX,4                 ; setup loop counter
          MOV BX, OFFSET dataIN    ; setup data pointer
          MOV AX,0                 ; initilaize AL
L1:       ADD AX,[BX]
          INC BX                   ; make BX point to next data item
          INC BX
          DEC CX                   ; decrement loop counter
          JNZ L1
          MOV sum,AX               ; load result into sum

          MOV AH,4CH    ;DOS: terminate program
          MOV AL,0      ;return code will be 0
          INT 21H       ;terminate the program

END start
```

Copy an Array

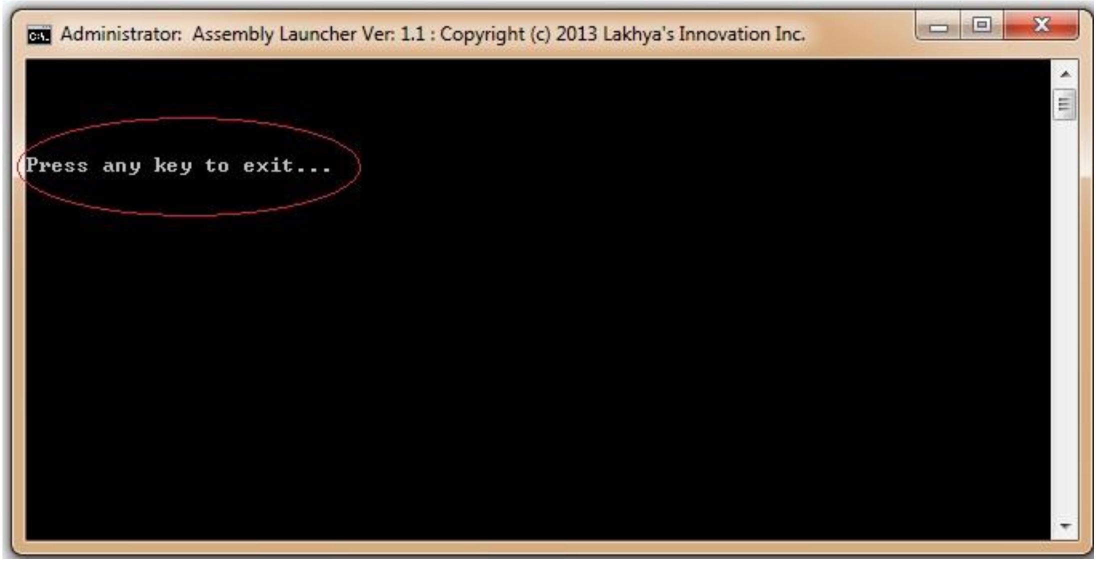
```
.MODEL SMALL
.STACK 100H

.DATA
        ORG 200H
dataIn  DB 'Q','W','E','R','T','Y'
copy    DB 6 DUP (?)
        DB '$'

.CODE
start:
        MOV AX,@DATA     ;set DS to point to the data segment
        MOV DS,AX

        MOV SI,OFFSET dataIn
        MOV DI,OFFSET copy
        MOV CX,6
movL:   MOV AL,[SI]
        MOV [DI],AL
        INC SI
        INC DI
        LOOP movL

        MOV AH,4CH   ;DOS: terminate program
        MOV AL,0     ;return code will be 0
        INT 21H      ;terminate the program
END start
```

DOS Interrupt 21H

AH	Operation	Input Register(s)	Output
4C	Program Terminate	AL=return code	None
01	Character Input (with echo)	None	AL=char
07	Character Input (no echo)	None	AL=char
0A	Buffered Keyboard Input	DX=string offset	None
02	Character Output	DL=char	None
09	Display String	DX=string offset	None

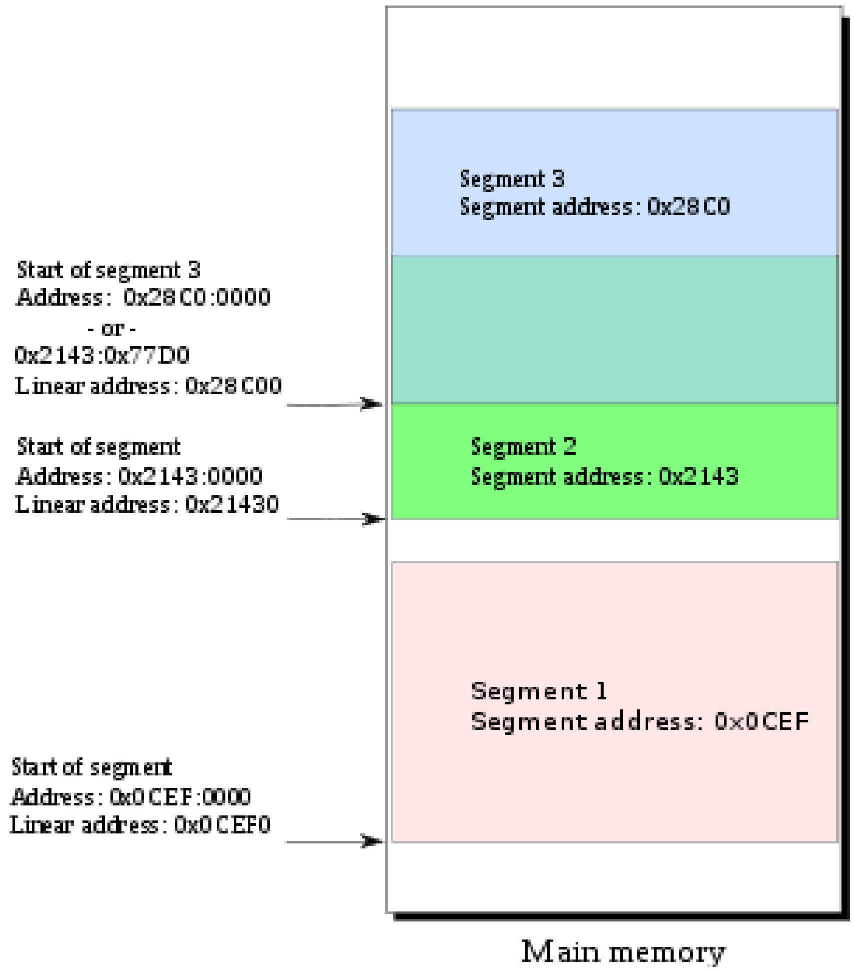
Program Termination

```
MOV AH,4CH    ; DOS: terminate program
MOV AL,0      ; return code will be 0
INT 21H       ; terminate the program
```

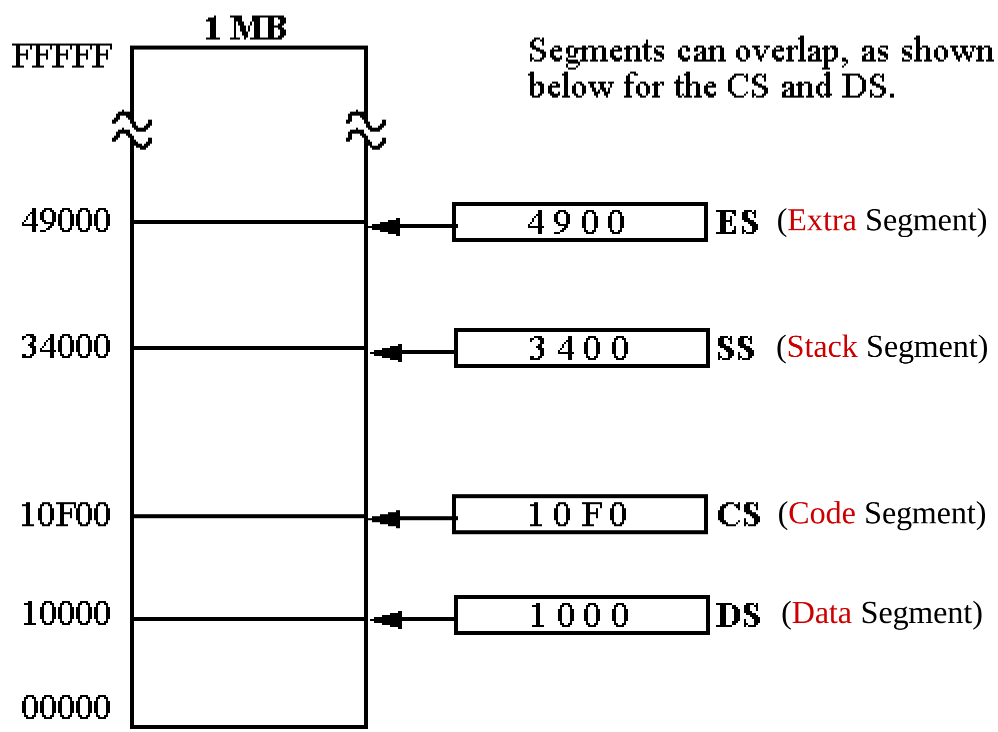


```
.MODEL SMALL
.STACK 100H
.DATA
TimePrompt        DB 'Is it after 12 noon (Y/N)?$'
MorningMsg        DB 13,10, 'Good morning, world!' ,13,10, '$'
AfternoonMsg      DB 13,10, 'Good afternoon, world!' ,13,10,'$'
DefaultMsg        DB 13,10,'Good day, world!' ,10,13, '$'
.CODE
start:            MOV AX,@data    ;set DS to point to the data segment
                  MOV DS,AX
                  LEA DX,TimePrompt    ;point to the time prompt
                  MOV AH,9    ;DOS: print string
                  INT 21H     ;display the time prompt
                  MOV AH,1    ;DOS: get character
                  INT 21H     ;get a single-character response
                  OR  AL, 20H ;force character to lower case
                  CMP AL,'y'  ;typed Y for afternoon?
                  JE IsAfternoon
                  CMP AL,'n'  ;typed N for morning?
                  JE IsMorning
                  LEA DX,DefaultMsg   ;default greeting
                  JMP DisplayG
IsAfternoon:      LEA DX,AfternoonMsg ;afternoon greeting
                  JMP DisplayG
IsMorning:        LEA DX,MorningMsg   ;before noon greeting
DisplayG:         MOV AH,9    ;DOS: print string
                  INT 21H     ;display the appropriate greeting
                  MOV AH,4cH  ;DOS: terminate program
                  MOV AL,0    ;return code will be 0
                  INT 21H     ;terminate the program
END start
```

Memory Structure

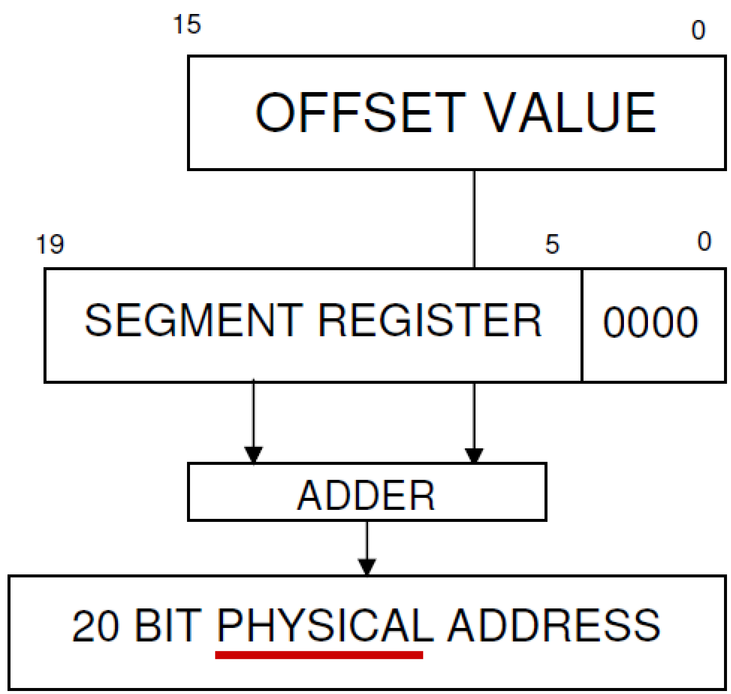


Segment Registers



Logical vs. Physical Address

"Segment:Offset" is the logical address



Endianness


8086
BIG ENDIAN - The way people always broke their eggs in the Lilliput land
LITTLE ENDIAN - The way the king then ordered the people to break their eggs

Endianness in 8086

```
MOV AX,35F3H      ; load 35F3H into AX
MOV [1500H],AX    ; copy contents of AX to offset 1500H
```

DS:1500 = F3

DS:1501 = 35

8086 Instructions

- Data movement instructions
  - move
  - push and pop
  - input/output operations
- Arithmetic, logic and shift instructions
- Decisions Making Instructions
  - conditional branches, unconditional jumps
  - calls & returns

Reg/Mem Data Movement

Instruction	Operation	Comments
`MOV dst,src`	dst ← src	
`XCHG src,dst`	dst ↔ src	
`LAHF`	AH ← flags1	
`SAHF`	flags1 ← AH	
`IN AL/AH/AX,port#` `IN AL/AH/AX,DX`	AL/AH/AX ← port# AL/AH/AX ← DX port	for port#<256 for port# >255
`OUT port#,AL/AH/AX` `OUT DX,AL/AH/AX`	port# ← AL/AH/AX DX port ← AL/AH/AX	for port#<256 for port# >255
`LEA dst,src`	dst ← EA(src)	Load Effective Address
`LDS reg,ptr`	reg(L) ← [ptr] reg(H) ← [ptr+1] DS(L) ← [ptr+2] DS(H) ← [ptr+3]	Load pointer using DS
`LES reg,ptr`	reg(L) ← [ptr] reg(H) ← [ptr+1] ES(L) ← [ptr+2] ES(H) ← [ptr+3]	Load pointer using ES
`XLAT`	AL ← memory byte DS:[BX + unsigned AL]	

Stack Manipulation

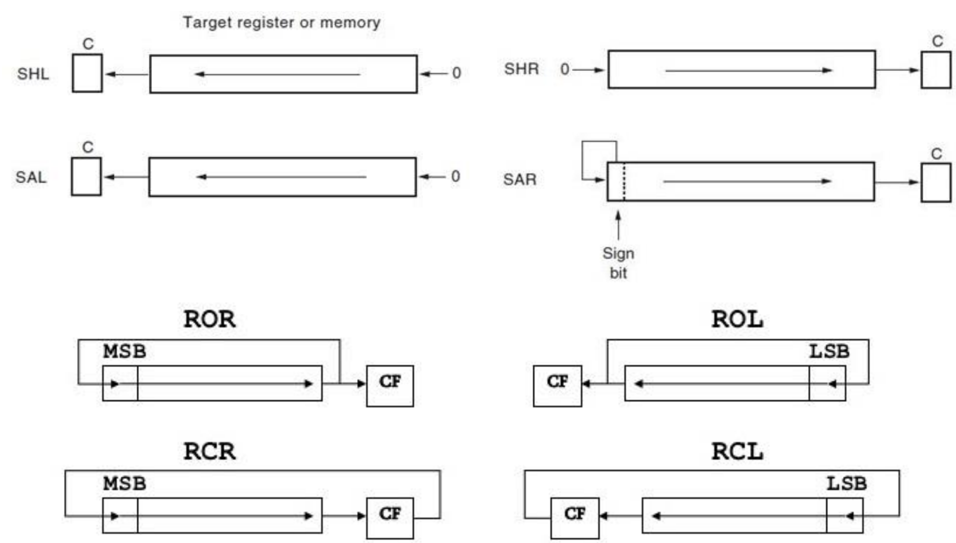
Instruction	Operation	Comments
`PUSH src`	SP ← SP-2 [SP] ← src(0-7) [SP+1] ← src(8-15)	Push src into stack
`POP dst`	dst(0-7) ← [SP] dst(8-15) ← [SP+1] SP ← SP+2	Pop dst out of stack
`PUSHF`	SP ← SP-2 [SP,SP+1] ← flags	Push Flag Register into stack
`POPF`	flags ← [SP,SP+1] SP ← SP+2	Pop Flag Register out of stack

Logical Instructions

Instruction	Operation
`NOT dst`	dst ← !dst
`AND dst,src`	dst ← dst ∧ src
`OR dst,src`	dst ← dst ∨ src
`XOR dst,src`	dst ← dst XOR src
`TEST dst,src`	dst ∧ src, update flags

Shift Instructions

Instruction	Operation	Comments
`SAL dst,cnt`	Shift dst to left cnt times	Arithmetic/Logic Shift Left
`SHL dst,cnt`		
`SAR dst,cnt`	Shift dst to right cnt times	Arithmetic Shift Right
`SHR dst,cnt`		Logic Shift Right
`RCL dst,cnt`	Rotate dst to left cnt times	Rotate with carry
`RCR dst,cnt`	Rotate dst to right cnt times	
`ROL dst,cnt`	Rotate dst to left cnt times	Rotate dst without carry
`ROR dst,cnt`	Rotate dst to right cnt times	



Arithmetic Instructions

Instruction	Operation	Comments
`ADD dst,src`	dst ← dst+src	Add
`ADC dst,src`	dst ← dst+src+CF	Add with carry
`SUB dst,src`	dst ← dst-src	Subtract
`SBB dst,src`	dst ← dst-(src+CF)	Subtract with borrow
`INC dst`	dst ← dst+1	Increment
`DEC dst`	dst ← dst-1	Decrement
`NEG dst`	dst ← 0-dst	Negate
`CMP dst,src`	dst-src, update flags	Compare
`MUL src`	AX ← AL*src DX:AX← AX*src	8 bit src 16 bit src
`IMUL src`	AX ← AL*src DX:AX← AX*src	8 bit signed src 16 bit signed src
`DIV src`	AL ← AL/src, AH ← AL%src AX ← DX:AX/src, DX ← DX:AX%src	8 bit src 16 bit src
`IDIV src`	AL ← AL/src, AH ← AL%src AX ← DX:AX/src, DX ← DX:AX%src	8 bit signed src 16 bit signed src

Arithmetic Adjust Instructions

Instruction	Comments
DAA	Decimal Adjust for Add
DAS	Decimal Adjust for Subtract
AAA	ASCII Adjust for Add
AAS	ASCII Adjust for Subtract
AAM	ASCII Adjust for Multiply
AAD	ASCII Adjust for Division
CBW	Convert Byte to Word
CWD	Convert Word to Double Word

Unsigned Multiplication/Division

`MUL src`	AX ← AL*src DX:AX← AX*src	8 bit src 16 bit src
`DIV src`	AL ← AL/src, AH ← AL%src AX ← DX:AX/src, DX ← DX:AX%src	8 bit src 16 bit src

Multiplication	Operand 1	Operand 2	Result
Byte × Byte	AL	Register or Memory	AX
Word × Word	AX	Register or Memory	DX,AX
Word × Byte	AL=Byte, AH=0	Register or Memory	DX,AX

Division	Numerator	Denominator	Quotient	Rem
Byte / Byte	AL=byte, AH=0	Register or Memory	AL	AH
Word / Word	AX=word, DX=0	Register or Memory	AX	DX
Word / Byte	AX=word	Register or Memory	AL	AH
DWord / Word	DX,AX=DWord	Register or Memory	AX	DX

Print a 3-digit Number

```
.DATA
num       DB 123
numSTR    DB "000$"

.CODE
start:
          MOV AX,@DATA      ; set DS to point to the data segment
          MOV DS,AX

          MOV AH,0          ; clear AH
          MOV AL,num        ; move the number into AL
          LEA SI,numSTR     ; move offset of numSTR into SI
          ADD SI,2          ; point to the end of string
L2:       MOV CL,10         ; move the divisor into CL
          DIV CL            ; divide AX to 10
          MOV DL,AH         ; move mod into DL
          ADD [SI],DL       ; store ASCII code of DL
          DEC SI            ; point to the previous character
          MOV AH,0          ; clear AH
          CMP AL,0          ; compare quotient with 0
          JNZ L2            ; repeat the loop if not zero

          LEA DX,numSTR     ; point to the num string
          MOV AH,9          ; DOS: print string
          INT 21H

          MOV AH,4CH    ; DOS: terminate program
          MOV AL,0      ; return code will be 0
          INT 21H       ; terminate the program
END start
```

Signed Multiplication/Division

`IMUL` for signed multiplication

`IDIV` for signed division

`CBW` (convert Byte to Word)

`CWD` (convert Word to Double word)

Multiplication	Operand 1	Operand 2	Result
Byte × Byte	AL	Register or Memory	AX
Word × Word	AX	Register or Memory	DX,AX
Word × Byte	AL=Byte, CBW	Register or Memory	DX,AX

Division	Numerator	Denominator	Quotient	Rem
Byte / Byte	AL=byte, CBW	Register or Memory	AL	AH
Word / Word	AX=word, CWD	Register or Memory	AX	DX
Word / Byte	AX=word	Register or Memory	AL	AH
DWord / Word	DX,AX=DWord	Register or Memory	AX	DX

Finding the Average

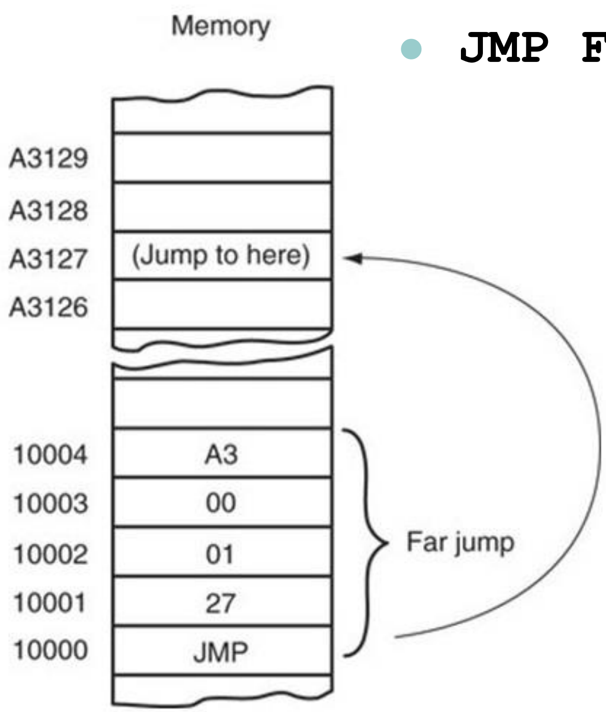
```
.MODEL SMALL
.STACK 100H
.DATA
dataIN   DB  -1,1,2,-2,0
         EVEN
sum      DW  ?
avg      DW  ?
.CODE
start:
         MOV AX,@DATA    ; set DS to point to the data segment
         MOV DS,AX
         MOV CX,5                   ; setup loop counter
         MOV SI, OFFSET dataIN      ; setup data pointer
         SUB BX,BX                  ; initilaize BX
AGAIN:   MOV AL,[SI]                ; move byte to AL
       ★ CBW                        ; extend sign
         ADD BX,AX
         INC SI                     ; make SI point to next data item
         DEC CX                     ; decrement loop counter
         JNZ AGAIN
         MOV sum,BX                 ; load result into sum
         MOV AX,sum                 ; load sum in AX
       ★ CWD                        ; extend sign
         MOV CX,5
         IDIV CX                       ; divide DX:AX to CX
         MOV avg,AX                 ; move quotient to avg
         MOV AH,4CH   ;DOS: terminate program
         MOV AL,0     ;return code will be 0
         INT 21H      ;terminate the program
END start
```

Decisions Making Instructions

- Unconditional Jump
- Conditional Jump
- Loop Control
- Subroutine Call/Return
- Software Interrupts

Unconditional Jump (JMP)

- Jump inside the segment (near jump)
  - `JMP label ;label in the same segment`
- Jump outside the segment (far jump)
  - `JMP FAR PTR label ;label=A300:0127`



Conditional Jumps

Mnemonic	Condition Tested	"Jump IF ..."
JA/JNBE	(CF = 0) and (ZF = 0)	above/not below nor zero
JAE/JNB	CF = 0	above or equal/not below
★ JB/JNAE	CF = 1	below/not above nor equal
JBE/JNA	(CF or ZF) = 1	below or equal/not above
JC	CF = 1	carry
JE/JZ	ZF = 1	equal/zero
JG/JNLE	((SF xor OF) or ZF) = 0	greater/not less nor equal
JGE/JNL	(SF xor OF) = 0	greater or equal/not less
★ JL/JNGE	(SF xor OF) = 1	less/not greater nor equal
JLE/JNG	((SF xor OF) or ZF) = 1	less or equal/not greater
JNC	CF = 0	not carry
JNE/JNZ	ZF = 0	not equal/not zero
JNO	OF = 0	not overflow
JNP/JPO	PF = 0	not parity/parity odd
JNS	SF = 0	not sign
JO	OF = 1	overflow
JP/JPE	PF = 1	parity/parity equal
JS	SF = 1	sign

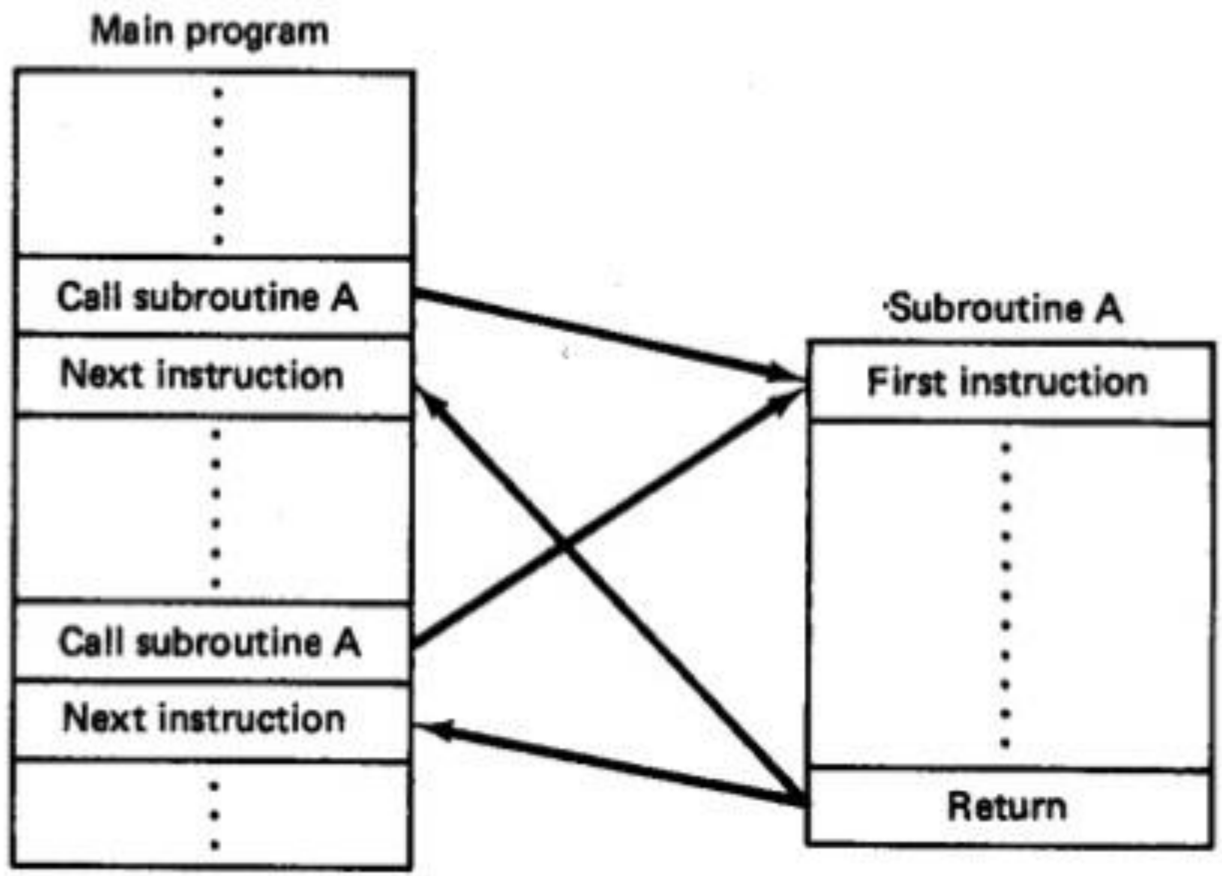
unsigned

signed

Short Jump

- All conditional jumps are short
- Conditional jump is a 2-bytes instruction:
  - jump operation code
  - relative address of jump target
- Target should be within [-128..127] bytes distance from IP

Call & Return from Subroutine



Near Procedure (Example)

```
.MODEL SMALL
.STACK 100H

.DATA
        ; place your data here

.CODE
Start:
        ; ...
        CALL SBN     ; call SBN
        ; ...

        ;-------------------------------------
        ; subroutine SBN
SBN     PROC NEAR
        ; ...
        RET          ; return from subroutine
SBN     ENDP

END     Start
```

Parameter Passing

- via Registers
  - Put parameter values in registers
- via Memory
  - Use the same variable names in the subroutine
- via Stack
  - Put parameter values in stack

Parameter Passing via Register

```
; Add A & B and put the result in C
; Pass parameters in registers
STSEG     SEGMENT STACK 'stack'
    DB   100H dup (?)
STSEG     ENDS
DTSEG SEGMENT
    A          DW 2      ; 1st operand
    B          DW 4      ; 2nd operand
    C          DW ?      ; C=A+B
DTSEG ENDS
CDSEG SEGMENT
Start:
        ASSUME DS:DTSEG,CS:CDSEG,SS:STSEG
        MOV AX,DTSEG     ; set DS to point to the data segment
        MOV DS,AX
        MOV AX,A
        MOV BX,B
        MOV CX,0
        CALL SB1      ; call subroutine SB1
        MOV C,CX

SB1     PROC NEAR
        PUSHF
        MOV CX,AX     ; CX <- A
        ADD CX,BX     ; CX <- A+B
        POPF
        RET
SB1     ENDP
CDSEG ENDS
END     Start
```

Parameter Passing via Memory

```
; Add A & B and put the result in C
; Pass parameters in Memory
STSEG     SEGMENT STACK 'stack'
    DB   100H dup (?)
STSEG     ENDS
DTSEG SEGMENT
    A         DW 2      ; 1st operand
    B         DW 4      ; 2nd operand
    C         DW 0      ; C=A+B
DTSEG ENDS
CDSEG SEGMENT
Start:
        ASSUME DS:DTSEG,CS:CDSEG,SS:STSEG
        MOV AX,DTSEG      ; set DS to point to the data segment
        MOV DS,AX
        CALL SB2          ; call subroutine SB2

SB2     PROC NEAR
        PUSHF
        PUSH AX
        MOV AX,A      ; AX <- A
        ADD AX,B      ; AX <- A+B
        MOV C,AX      ; C <- AX
        POP AX
        POPF
        RET
SB2     ENDP
CDSEG ENDS
END     Start
```

Parameter Passing via Stack - 1

```
; Add A & B and put the result in C
; Pass parameters via Stack
STSEG     SEGMENT STACK 'stack'
     DB   100H dup (?)
STSEG     ENDS
DTSEG SEGMENT
     A          DW 2   ; 1st operand
     B          DW 4   ; 2nd operand
     C          DW 0   ; C=A+B
DTSEG ENDS
CDSEG SEGMENT
Start:
          ASSUME DS:DTSEG,CS:CDSEG,SS:STSEG
          MOV AX,DTSEG
          MOV DS,AX
          PUSH C
          PUSH B
          PUSH A
          CALL SB3      ; call subroutine SB1
          POP BX        ; BX <- A
          POP BX        ; BX <- B
          POP BX        ; BX <- C   (C is updated by sub3)
          MOV C,BX
```

Parameter Passing via Stack - 2

```
SB3       PROC NEAR
          PUSHF
          PUSH AX
          PUSH BX
          PUSH BP
          MOV BP,SP
          MOV AX,SS:[BP+10]    ; AX <- A the value of A is in the stack
          MOV BX,SS:[BP+12]    ; BX <- B the value of B is in the stack
          ADD AX,BX
          MOV SS:[BP+14],AX    ; C <- AX+BX update the value of C in the stack
          POP BP
          POP BX
          POP AX
          POPF
          RET
SB3       ENDP
CDSEG ENDS
END       Start
```

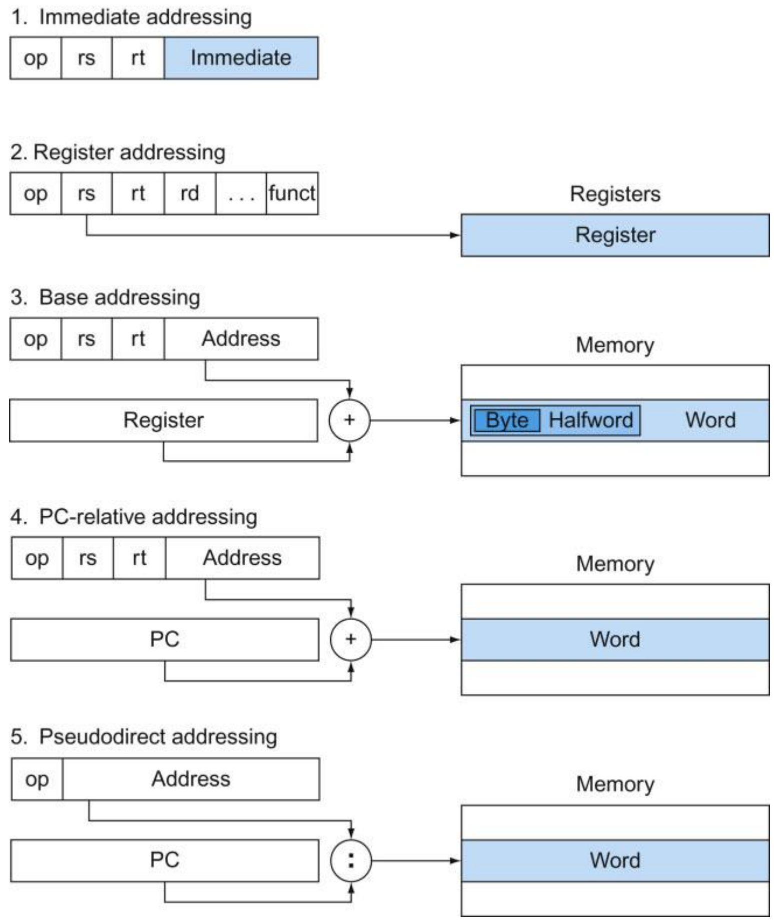
CPU Control Instructions

Instruction	Operation	Comments
STC	CF ← 1	Set Carry Flag
CLC	CF ← 0	Clear Carry Flag
CMC	CF ← !CF	Complement Carry Flag
STD	DF ← 1	Set Direction Flag
CLD	DF ← 0	Clear Direction Flag
STI	IF ← 1	Set Interrupt Flag
CLI	IF ← 0	Clear Interrupt Flag
HLT		Halt
WAIT		Wait
LOCK		Lock
NOP		No Operation
ESC		Escape

Addressing Modes (in general)

- Implicit
- Immediate
- Register (direct)
- Register indirect
- Base or displacement addressing
- Indexed addressing
- Auto-increment / Auto-decrement
- PC-relative
- Memory direct
- Memory indirect

MIPS Addressing Modes



8086 Addressing Modes

Addressing Modes

- Immediate Addressing
- Register Addressing
  - Register Direct
  - Register Indirect
- Memory Direct
- Displacement Addressing
  - Based Addressing
  - Indexed Addressing
  - Based-Indexed Addressing
- Implied Addressing

Immediate Addressing

```
MOV   AX,2550H      ;move 2550H into AX
MOV   CX,625        ;load the decimal value 625 into CX
MOV   BL,40H        ;load 40H into BL
```

Register Reference

```
MOV   BX,DX      ;copy the contents of DX into BX
MOV   ES,AX      ;copy the contents of AX into ES
ADD   AL,BH      ;add the contents of BH to contents of AL
```

Direct

Indirect

```
MOV   AL,[BX]    ;moves into AL the contents of the memory location
                 ;pointed to by DS:BX.

MOV   CL,[SI]    ;move contents of DS:SI into CL
MOV   [DI],AH    ;move contents of AH into DS:DI
```

Memory Direct

```
MOV DL,[2400H]    ;move contents of DS:2400H to DL
```

Example:

Assuming DS=1512H, find physical memory address and its contents after executing the following code:

```
MOV AL,99H
MOV [3518H],AL
```

Based Relative

```
MOV    CX,[BX]+10        ;move DS:BX+10 and DS:BX+10+1 into CX
                         ;PA = DS (shifted left) + BX + 10
```

"MOV CX,10[BX]" یا "MOV CX,[BX+10]"

```
MOV    AL,[BP]+5         ;PA = SS (shifted left) + BP + 5
```

"MOV AL,5[BP]" یا "MOV AL,[BP+5]"

Indexed Relative

```
MOV DX,[SI]+5    ; PA=DS (shifted left) + SI + 5
MOV CL,[DI]+20   ; PA=DS (shifted left) + DI + 20
```

Based Indexed Relative

```
MOV   CL,[BX][DI]+8      ;PA = DS (shifted left) + BX + DI + 8
MOV   CH,[BX][SI]+20     ;PA = DS (shifted left) + BX + SI + 20
MOV   AH,[BP][DI]+12     ;PA = SS (shifted left) + BP + DI + 12
MOV   AH,[BP][SI]+29     ;PA = SS (shifted left) + BP + SI + 29
```

```
MOV   AH,[BP+SI+29]          MOV   AH,[SI+BP+29]
```

Summary

Addressing Mode	Operand	Default Segment
Register	reg	none
Immediate	data	none
Direct	[offset]	DS
Register indirect	[BX]	DS
	[SI]	DS
	[DI]	DS
Based relative	[BX]+disp	DS
	[BP]+disp	SS
Indexed relative	[DI]+disp	DS
	[SI]+disp	DS
Based indexed relative	[BX][SI]+disp	DS
	[BX][DI]+disp	DS
	[BP][SI]+ disp	SS
	[BP][DI]+ disp	SS

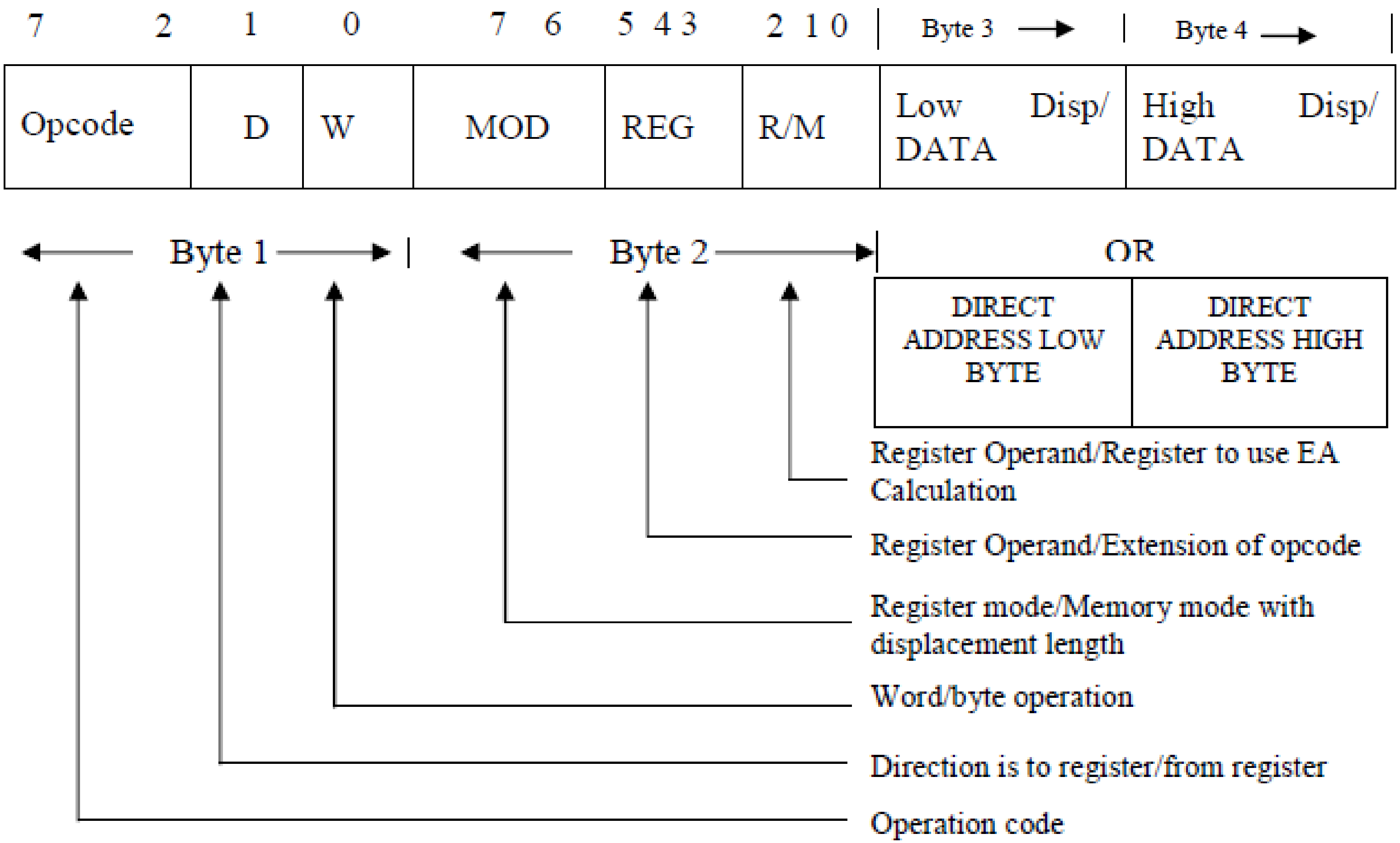
x86 Memory Addressing Modes

src/dst operand	2^{nd} src operand
Register	Register
Register	Immediate
Register	Memory
Memory	Register
Memory	Immediate

Memory addressing modes

- Address in register
- Address = R_{base} + displacement
- Address = $R_{base} + 2^{scale} \times R_{index}$ (scale = 0, 1, 2, or 3)
- Address = $R_{base} + 2^{scale} \times R_{index}$ + displacement

8086 Instruction Encoding



8086 Instruction Encoding (cont.)

MOD (2 bits)	Interpretation
00	Memory mode with no displacement follows except for 16 bit displacement when R/M=110
01	Memory mode with 8 bit displacement
10	Memory mode with 16 bit displacement
11	Register mode (no displacement)

REG	W=0	W=1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

R/M	W=0	W=1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

8086 Instruction Encoding (cont.)

R/M	MOD=00	MOD 01	MOD 10
000	(BX) + (SI)	(BX)+(SI)+D8	(BX)+(SI)+D16
001	(BX)+(DI)	(BX)+(DI)+D8	(BX)+(DI)+D16
010	(BP)+(SI)	(BP)+(SI)+D8	(BP)+(SI)+D16
011	(BP)+(DI)	(BP)+(DI)+D8	(BP)+(DI)+D16
100	(SI)	(SI) + D8	(SI) + D16
101	(DI)	(DI) + D8	(DI) + D16
110	Direct address	(BP) + D8	(BP) + D16
111	(BX)	(BX) + D8	(BX) + D16

8086 Instruction Encoding (example 1)

Example 1 : **<u>Code for</u>** MOV CH, BL

This instruction transfers 8 bit content of BL into CH

The 6 bit Opcode for this instruction is 100010_2 D bit indicates whether the register specified by the REG field of byte 2 is a source or destination operand.

D=0 indicates BL is a source operand.
W=0 byte operation
In byte 2, since the second operand is a register MOD field is 11_2.
The R/M field = 101 (CH)
Register (REG) field = 011 (BL)
Hence the machine code for MOV CH, BL is
10001000 11 011 101
Byte 1 Byte2
= 88DDH

7 2	1	0	7 6	5 4 3	2 1 0	Byte 3 →	Byte 4 →
Opcode	D	W	MOD	REG	R/M	Low Disp/ DATA	High Disp/ DATA

8086 Instruction Encoding (example 2)

Example 2: Code for SUB BX, (DI)

This instruction subtracts the 16 bit content of memory location addressed by DI and DS from Bx.
The 6 bit Opcode for SUB is 001010_2.
D=1 so that REG field of byte 2 is the destination operand. W=1 indicates 16 bit operation.
MOD = 00
REG = 011
R/M = 101

The machine code is	0010	1011	0001	1101
	2	B	1	D

7 2	1	0	7 6	5 4 3	2 1 0	Byte 3 →	Byte 4 →
Opcode	D	W	MOD	REG	R/M	Low Disp/ DATA	High Disp/ DATA

8086 Instruction Encoding (example 4)

Example 4 : Code for MOV DS : 2345 [BP], DX

Here we have to specify DX using REG field. The D bit must be 0, indicating that Dx is the source register. The REG field must be 010 to indicate DX register. The w bit must be 1 to indicate it is a word operation. 2345 [BP] is specified with MOD=10 and R/M = 110 and displacement = 2345 H.

Whenever BP is used to generate the Effective Address (EA), the default segment would be SS. In this example, we want the segment register to be DS, we have to provide the **segment override prefix** byte (SOP byte) to start with. The SOP byte is **001 xx 110**, where SR value is provided as per table shown below.

xx	**Segment register**
00	ES
01	CS
10	SS
11	DS

To specify DS register, the SOP byte would be 001 11 110 = 3E H. Thus the 5 byte code for this instruction would be 3E 89 96 45 23 H.

SOP	**Opcode**	**D**	**W**	**MOD**	**REG**	**R/M**	**LB disp.**	**HD disp.**
3EH	1000 10	0	1	10	010	110	45	23

Intel 80x86 Architecture

- Intel 80x86 CISC Architecture
  - Supports many addressing modes
  - Supports complicated instructions
    - Reference manuals more than thousand pages
  - But its performance is no worse than RISC architectures (if not better)
    - e.g. Apple recently switched from PowerPC to Intel
  - Why? Microarchitecture
    - Translates CISC instructions into RISC ones in hw

Implementing IA-32

- RISC & CISC Hybrid Solution
- Complex instruction set makes implementation difficult
  - HW translates instructions to simpler micro-operations
    - Simple instructions: 1–1
    - Complex instructions: 1–many
  - Micro-engine similar to RISC
- Comparable performance to RISC
  - Compilers avoid complex instructions

RISC & CISC

- Hybrid Solution
  - RISC core & CISC interface
  - Taking advantage of both architectures

RISC / CISC Debate

8086/88 ISA

- History
- ISA Concerns:
  - Memory models/ Registers/ Addressing modes
- Instruction Set
  - Move, Arithmetic, Logic, Shift, Control Transfer, …
- Instruction Encoding
- 8086 Internal Architecture

"XCHG" Example

```
; Demonstrate the application of XCHG instruction

.MODEL SMALL

.STACK 100H

.DATA
data1    DB 'A'
data2    DB 'B'

.CODE
Start:
         MOV AX,@DATA     ; set DS to point to the data segment
         MOV DS,AX

         MOV AL,data1     ; mov contents of data1 into AL
         XCHG AL,data2    ; exchange contents of AL and data2
         MOV data1,AL     ; mov new contents of AL into data2


         MOV AH,4CH   ; DOS: terminate program
         MOV AL,0     ; return code will be 0
         INT 21H      ; terminate the program
END Start
```

Lookup Table Example

```
.DATA
ATAB      DB '0','1','2','3','4','5','6','7'
          DB '8','9','A','B','C','D','E','F'
HexV      DB 10
ASCV      DB ?

.CODE
start:
          MOV AX,@DATA      ;set DS to point to the data segment
          MOV DS,AX

          LEA BX,ATAB       ; mov table offset into BX
          MOV AL,HexV       ; mov the hex data into AL
          XLAT              ; get the ASCII equivalent
          MOV ASCV,AL       ; mov it to memory

          MOV DL,ASCV
          MOV AH,2      ; DOS: print char
          INT 21H       ; display result


          mov AH,4CH    ; DOS: terminate program
          mov AL,0      ; return code will be 0
          int 21H       ; terminate the program
END start
```

Stack Example

```
; This program evaluates Q=A+(B-C)
; in the form of Q=ABC-+ using stack

STSEG   SEGMENT STACK 'stack'
        DB 100H DUP (?)
STSeg   ENDS

DTSeg   Segment
A    DW 3
B    DW 8
C    DW 6
DTSeg   ENDS

CDSeg   Segment
        ASSUME CS:CDSeg,DS:DtSeg,SS:StSeg
start:
        MOV AX,DTSeg     ; set DS to point to the data segment
        MOV DS,AX
        MOV AX,STSeg     ; set DS to point to the stack segment
        MOV SS,AX

        PUSH A
        PUSH B
        PUSH C
        POP CX        ; pop C into CX
        POP BX        ; pop B into BX
        SUB BX,CX     ; BX=B-C
        PUSH BX
        POP CX        ; pop B-C into CX
        POP BX        ; pop A into BX
        ADD BX,CX     ; BX=A+(B-C)
        PUSH BX

        MOV AH,4CH    ; DOS: terminate program
        MOV AL,0      ; return code will be 0
        INT 21H       ; terminate the program
CDSeg   ENDS
END Start
```

Convert to Capital Letters

Letter	ASCII Code	Letter	ASCII Code
A	01000001	a	01100001
B	01000010	b	01100010
C	01000011	c	01100011
...	...		...
Y	01011001	y	01111001
Z	01011010	z	01111010

```
.DATA
data1      DB "mY naME Is joEz"
data2      DB 15 DUP (?)
           DB '$'

.CODE
start:
           MOV AX,@DATA      ;set DS to point to the data segment
           MOV DS,AX

           MOV SI,OFFSET data1
           MOV DI,OFFSET data2
           MOV CX,15
L1:        MOV AL,[SI]
           CMP AL,'a'
           JB  OVER              ; no need to convert
           CMP AL,'z'
           JA  OVER              ; no need to convert
           AND AL,11011111B      ; mask D5 to convert to uppercase
OVER:      MOV [DI],AL           ; copy the letter back
           INC SI
           INC DI
           LOOP L1

           MOV AH,4CH    ;DOS: terminate program
           MOV AL,0      ;return code will be 0
           INT 21H       ;terminate the program
END start
```

Check Parity Flag

				OF	DF	IF	TF	SF	ZF	0	AF	1	PF	1	CF

```
; Check Parity Flag
.MODEL SMALL
.STACK 100H

.DATA
ParityF      DB 'P'
NoParity     DB 'N'

.CODE
Start:
        MOV AX,@DATA        ; set DS to point to the data segment
        MOV DS,AX

        SUB AL,AL           ; force parity flag to 1
        INC AL              ; force parity flag to 0
        LAHF                ; load flag reg into AH
        MOV DL,NoParity     ; suppose there is no parity
        SHR AH,3            ; shift parity flag into CF
        JNC NEXT
        MOV DL,ParityF
NEXT:   MOV AH,2        ;DOS: print char
        INT 21H         ;display result
        MOV AH,4CH      ; DOS: terminate program
        MOV AL,0        ; return code will be 0
        INT 21H         ; terminate the program
END Start
```

Find Maximum Number in a List

```
DtSeg    Segment
dataIn   DB 12,23,1,45,26
max      DB ?
DtSeg    ENDS

CDSeg    Segment
         ASSUME CS:CDSeg,DS:DtSeg,SS:StSeg
start:
         MOV AX,DtSeg     ; set DS to point to the data segment
         MOV DS,AX

         MOV CX,5
         MOV BX,OFFSET dataIn
         MOV AL,0
L1:      CMP AL,[BX]
         JA  NEXT     ; continue to search if AL is already greater
         MOV AL,[BX] ; update AL
Next:    INC BX
         LOOP L1
         MOV max,AL

         MOV AH,4CH   ; DOS: terminate program
         MOV AL,0     ; return code will be 0
         INT 21H      ; terminate the program
CDSeg   ENDS
END start
```

Character Input

```
.DATA
inChar  DB ?


MOV AH,01          ; move option (01) to AH
INT 21H            ; input character (with echo)
MOV inChar, AL     ; move the input char to inChar


MOV AH,07          ; move option (01) to AH
INT 21H            ; input character (no echo)
MOV inChar, AL     ; move the input char to inChar
```

String Input

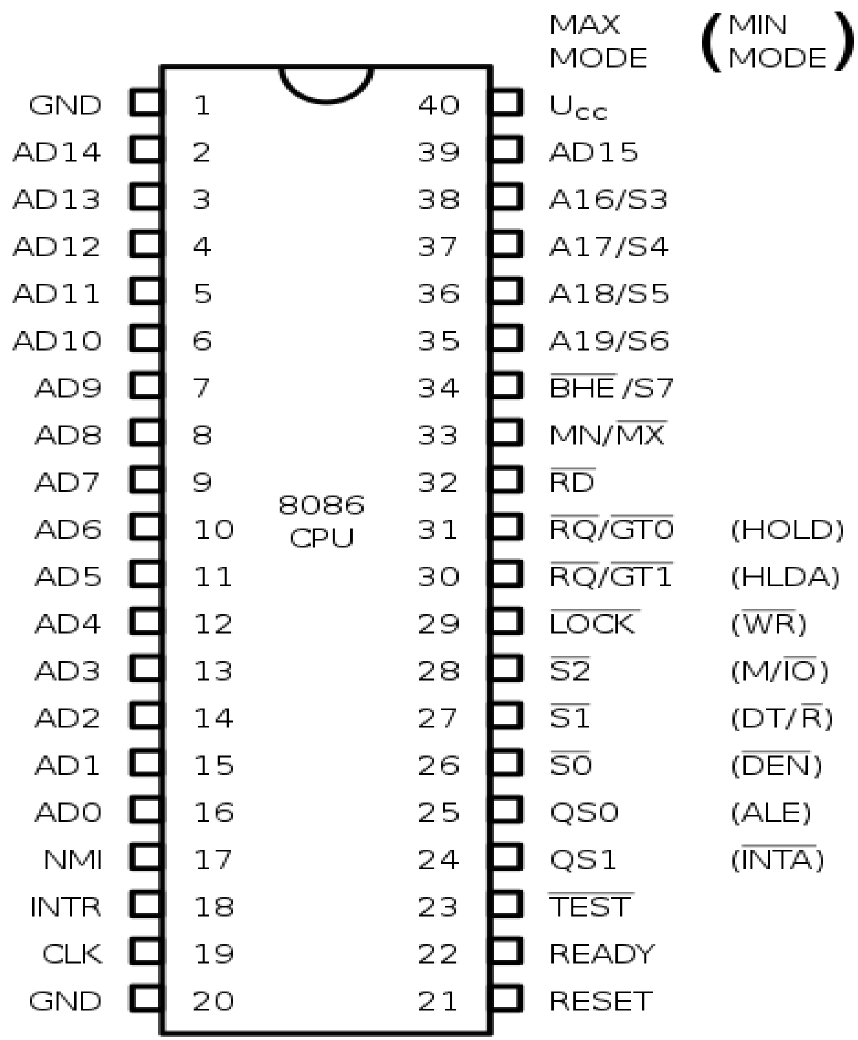
```
.DATA
inBuf   Label BYTE         ; input buffer
Bsize   DB 10              ; buffer size
Rsize   DB ?               ; real size
inStr1  DB 10 DUP ' '      ; input string
```

```
LEA DX,inBuf        ; move buffer offset to DX
MOV AH,0AH          ; move option (0AH) to AH
INT 21H             ; input string

LEA BX,inStr1       ; move string offset to BX
MOV CL,Rsize        ; move real buffer size to CL
SUB CH,CH           ; clear CH
MOV SI,CX           ; move index of CR to SI
MOV BYTE PTR[BX+SI],'$' ; replace CR with $

LEA DX,inStr1       ; move string offset to DX
MOV AH,09           ; move option (09) to AH
INT 21H             ; display string
```

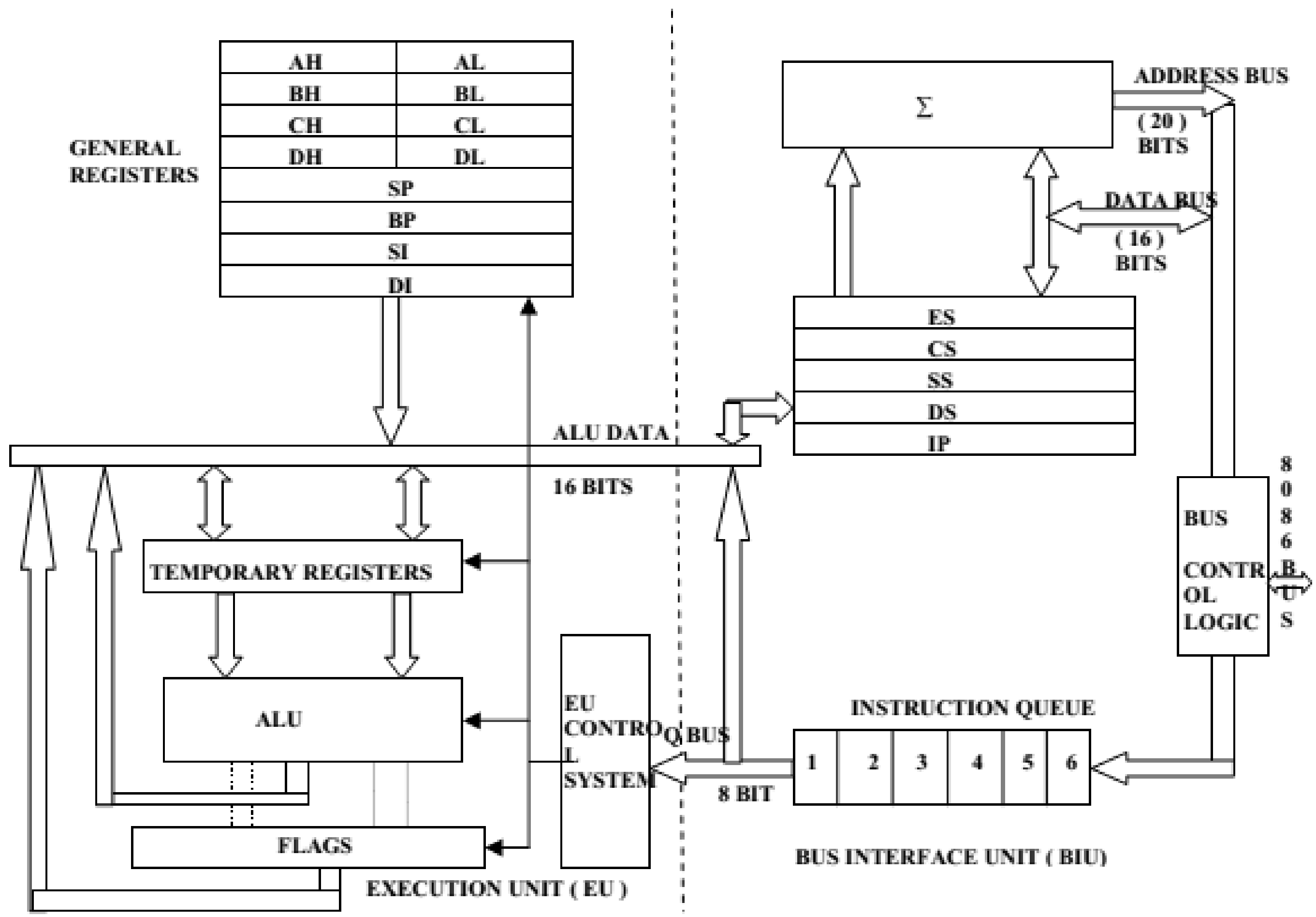
Character/ String Output

```
.DATA
CR         EQU 13
LF         EQU 10
outStr DB "Have a nice day $"

MOV DL,CR          ; move the character to be displayed
MOV AH,02          ; move option (02) to AH
INT 21H            ; display character

MOV DL,LF          ; move the character to be displayed
MOV AH,02          ; move option (02) to AH
INT 21H            ; display character

LEA DX,outStr      ; move string offset to DX
MOV AH,09          ; move option (09) to AH
INT 21H            ; display string
```

8086 Chip


MAX MODE
(MIN MODE)
GND 1
AD14 2
AD13 3
AD12 4
AD11 5
AD10 6
AD9 7
AD8 8
AD7 9
AD6 10
AD5 11
AD4 12
AD3 13
AD2 14
AD1 15
AD0 16
NMI 17
INTR 18
CLK 19
GND 20
8086 CPU
40 Ucc
39 AD15
38 A16/S3
37 A17/S4
36 A18/S5
35 A19/S6
34 BHE /S7
33 MN/MX
32 RD
31 RQ/GT0 (HOLD)
30 RQ/GT1 (HLDA)
29 LOCK (WR)
28 S2 (M/IO)
27 S1 (DT/R)
26 S0 (DEN)
25 QS0 (ALE)
24 QS1 (INTA)
23 TEST
22 READY
21 RESET

8086 Internal Architecture



Bus Interface Unit (BIU)

- Takes care of all data and addresses transfers on the buses:
  - sending addresses
  - fetching instructions from the memory
  - reading data from the ports and the memory
  - writing data to the ports and the memory
- EU has no direction connection with system buses
- EU and BIU are connected with the internal bus

BIU Functional Parts

- Instruction queue:
  - Up to 6 bytes of next instructions is stored in the instruction queue
  - When EU executes instructions and is ready for its next instruction, then it reads the instruction from this instruction queue resulting in increased execution speed
  - Fetching the next instruction while the current instruction executes is called prefetching
- Segment registers (CS, DS, SS, ES)
- Instruction pointer:
  - A 16-bit register that holds the address of the next instruction to be executed

Execution Unit (EU)

- Telling the BIU from where to fetch the data
- Decode the instructions
  - using the instruction decoder
- Execute the instructions
  - using the ALU
- EU has no direct connection with system buses

EU Functional Parts

- ALU: Arithmetic and Logical Unit
- Flag Register
- General Purpose Registers:
  - AX: Accumulator Register
  - BX: Base Register
  - CX: Counter Register
  - DX: Data Register
  - SI/DI: Source/Destination Index
- Stack/Base Pointer Register