

Intel 8086 Processor

Mohsen Nickray
University of QOM

Intel Processors

Table 1-1: Evolution of Intel's Microprocessors

Product	8080	8085	8086	8088	80286	80386	80486
Year introduced	1974	1976	1978	1979	1982	1985	1989
Clock rate (MHz)	2 - 3	3 - 8	5 - 10	5 - 8	6 - 16	16 - 33	25 - 50
No. transistors	4500	6500	29,000	29,000	130,000	275,000	1.2 million
Physical memory	64K	64K	1M	1M	16M	4G	4G
Internal data bus	8	8	16	16	16	32	32
External data bus	8	8	16	8	16	32	32
Address bus	16	16	20	20	24	32	32
Data type (bits)	8	8	8, 16	8, 16	8, 16	8, 16, 32	8, 16, 32

Evolution from 8080/8085 to 8086

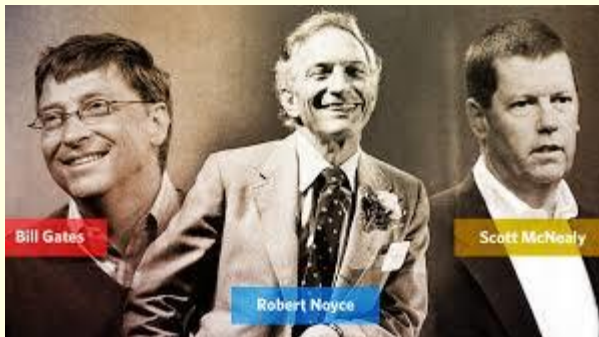
- The 8086 capacity of 1 megabyte of memory exceeded the 8080/8085's capability of handling a maximum of 64K bytes of memory.
- 8 bit in contrast to 16 bit processor
- The 8086 was a pipelined processor, as opposed the non pipelined 8080/8085

Evolution of 8086 to 8088

- The 8086 is a microprocessor with a 16 bit data bus internally and externally, meaning that all registers are 16 bits wide and there is a 16 bit data bus to transfer data in and out of the CPU. But
 - At that time all peripherals around the processor was 8 bit
 - In addition, a printed circuit board with a 16 bit data bus was much more expensive.
- Then 8088 *came out*, it is *identical* to 8086 *as far as programming* is concerned, but externally it has an 8-bit data bus

Success of 8088

- IBM picked up 8088 in designing IBM PC.
- IBM PC= Intel + Microsoft
- IBM and Microsoft made it an open system in contrast with apple as a close system.



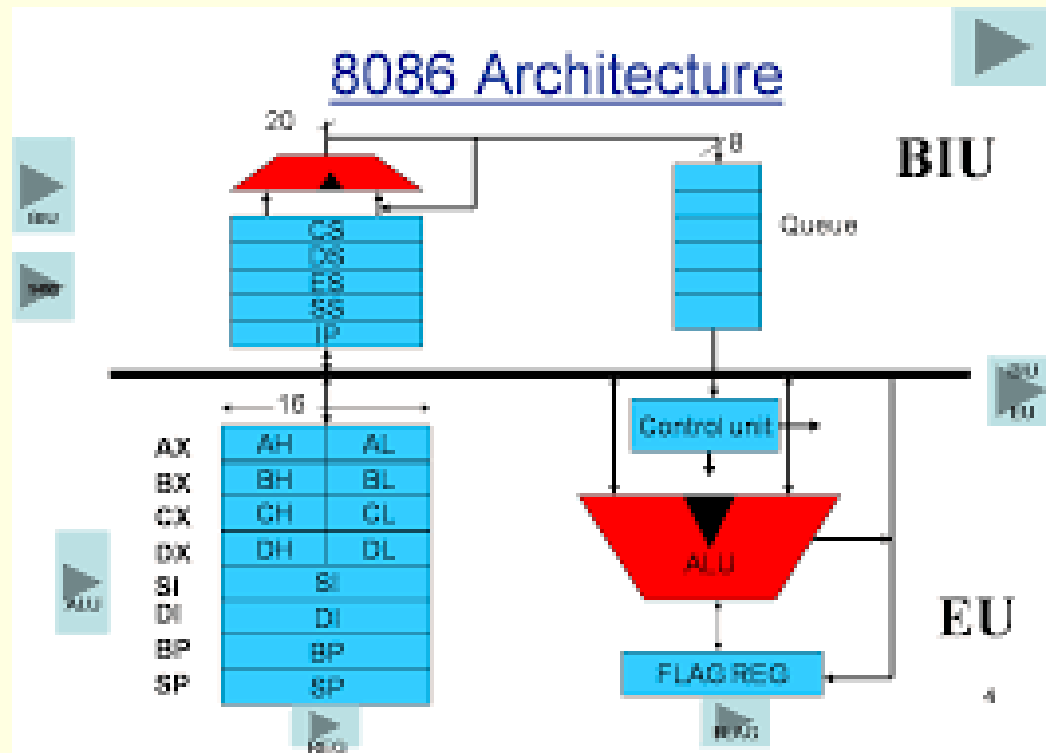
Other microprocessors

- 80286 in 1982
 - 16 bit internal and external data bus and 24 address lines
 - Operates in two modes: real and protected.
 - Virtual memory
- 80386 in 1985
 - Internally and externally 32 bit data bus and 32 address lines
- 80486 in 1989
 - Math coprocessor on a single chip
 - Some extra features like cache

8086 Architecture

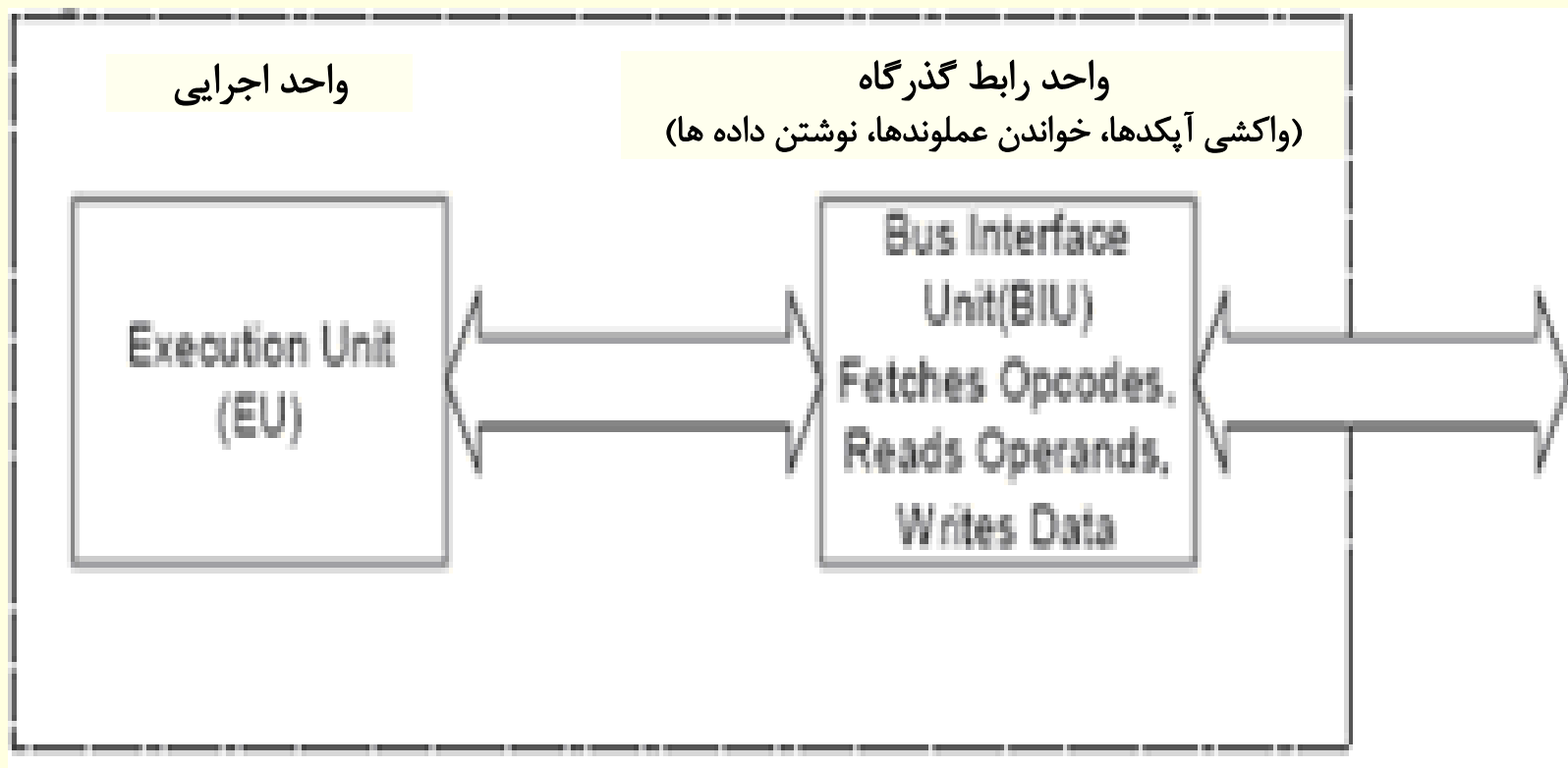


8086/ 8088 Architecture

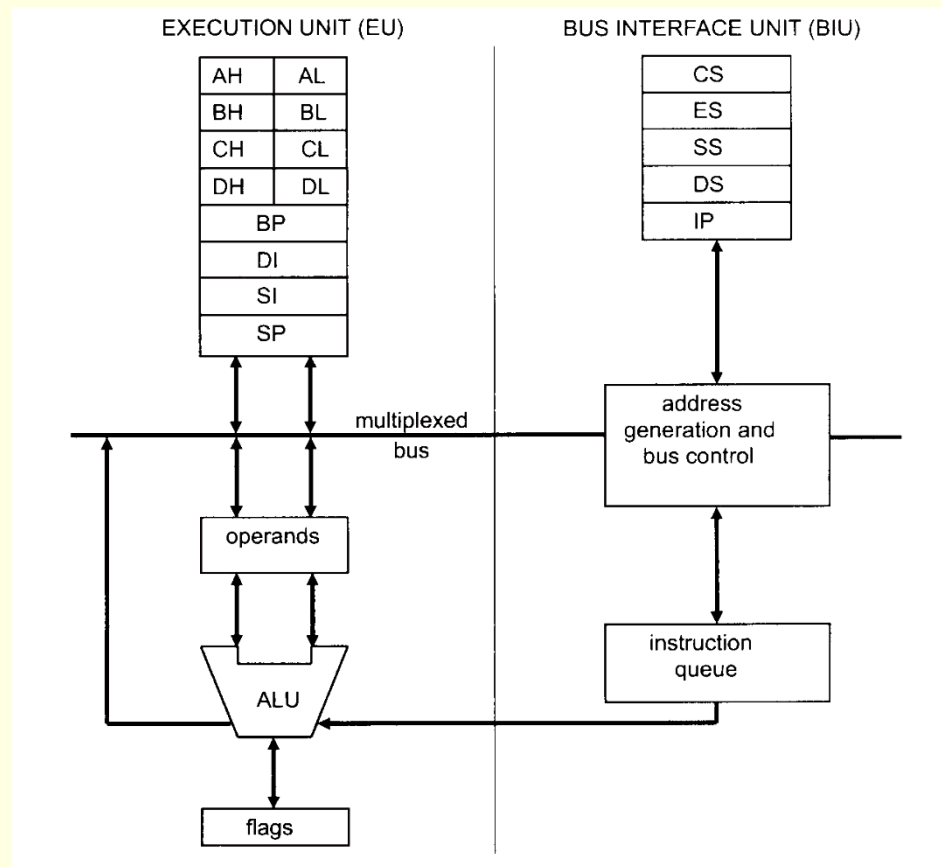


EU & BIU

Intel splitted the internal structure of 8086/8088 into two section.



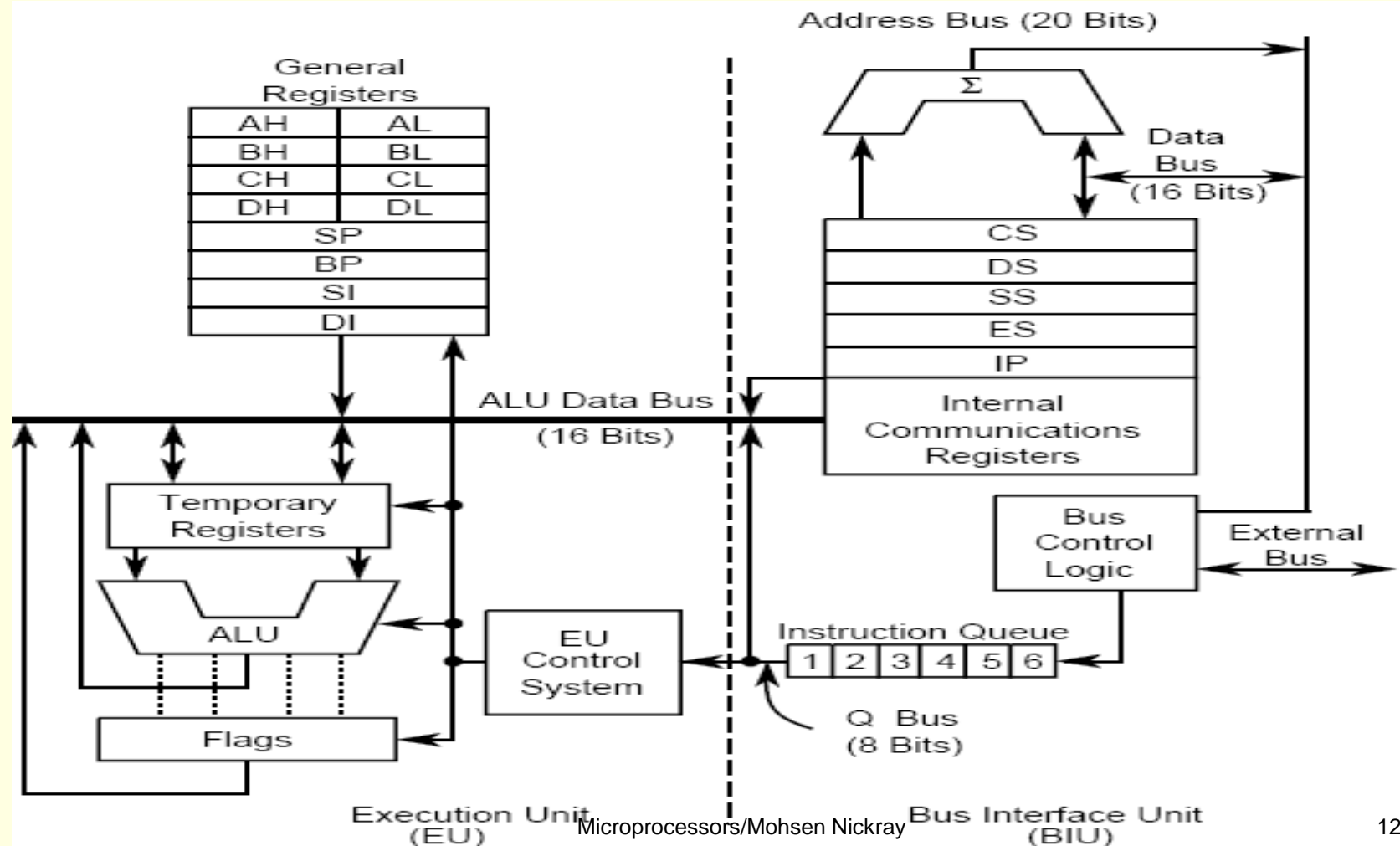
Big Picture of 8086/8088



EU and BIU

- These two sections work *simultaneously*.
- BIU accesses memory and peripherals while the EU executes instructions *previously* fetched.
- This works only if the BIU *keeps ahead* of the EU, thus the BIU of the 8086/8088 has a *buffer or queue*.
- The buffer is *4 bytes* long in 8088 and *6 bytes* in 8086.
- If any execution takes too long to execute, the *queue is filled*.

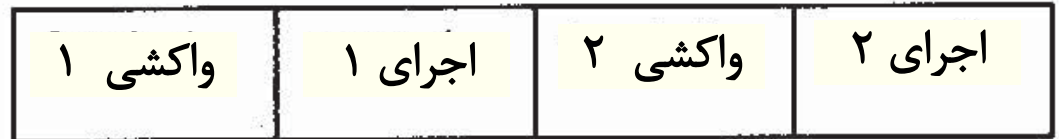
8086/8088 Architecture



Benefit of Splitting into EU and BIU

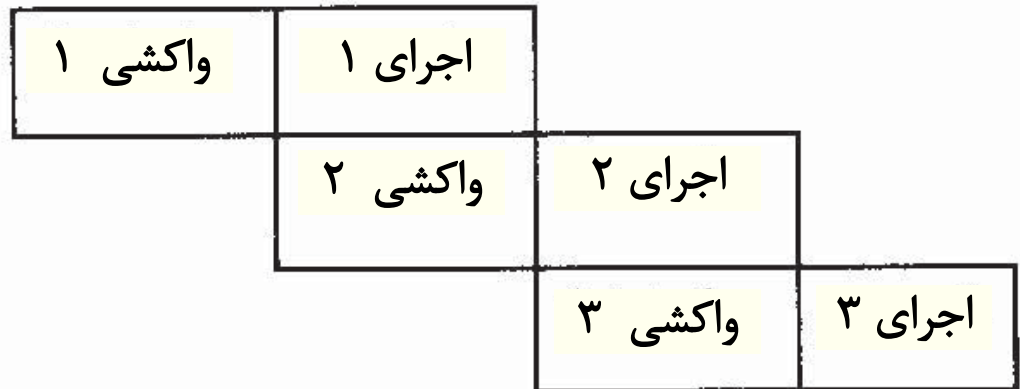
(پردازش غیر موازی)

nonpipelined
(e.g., 8085)



(پردازش موازی)

pipelined
(e.g., 8086)



واحد رابط گذرگاه (Bus Interface Unit)

- واحد BIU تشکیل شده از:
- صف جریان بایتهای دستورات
- مجموعه ای از Segment رجیسترها
- اشاره گر دستورات یا IP

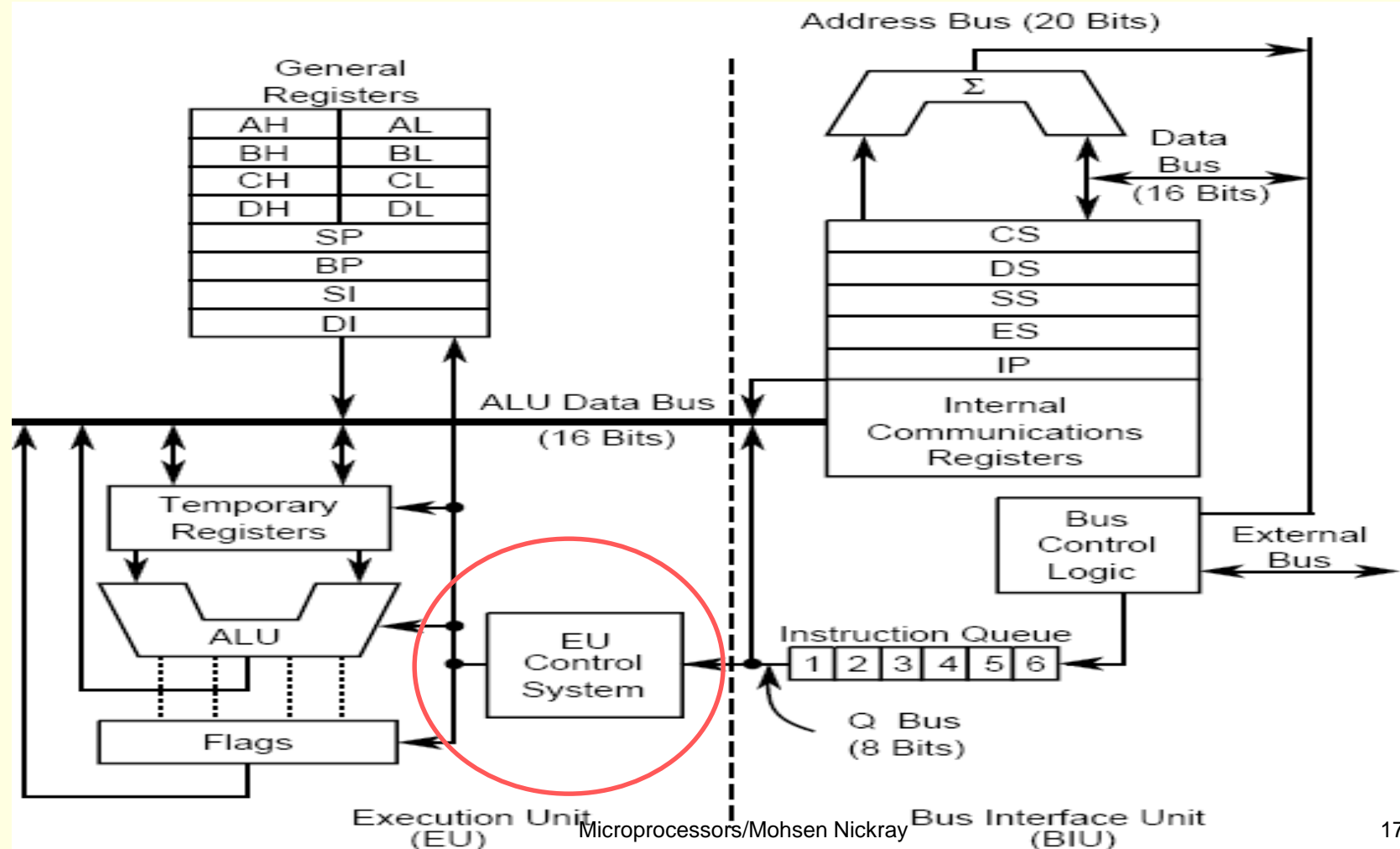
BIU – صف بایت دستورات

- دستورات پردازنده ۸۰۸۶ دارای طول ها متنوع از ۱ تا ۶ بایت هستند.
- بنابراین عمل واکشی و اجرای دستورات همزمان انجام میشوند تا بدین ترتیب کارایی ریزپردازنده افزایش یابد.
- واحد **BIU** جریان دستورات پشت سرهم را به واحد **EU** از طریق یک صف ۶ بایتی که بین این دو قرار دارد، وارد میکند.
- این صف میتواند بعنوان نوعی از اجرای خط لوله ای (البته بصورت خیلی ساده) که شامل دو مرحله واکشی و اجرا است، به کار رود.

BIU – صف بایت دستورات

- اجرا و دیکد کردن دستورات به باسها نیازی ندارد.
- درحین اینکه دستورات اجرا میشوند، واحد BIU دستور بعدی را که حداکثر ممکن است ۶ بایت باشد، واکشی کرده و درون ۶ بایت صف قرار میدهد. پس این صف شامل دستور بعدی میشود.
- زمانیکه واحد EU برای اجرای دستور بعدی آماده شد، EU به راحتی دستور بعدی را از صف (که در واحد BIU است) میخواند.

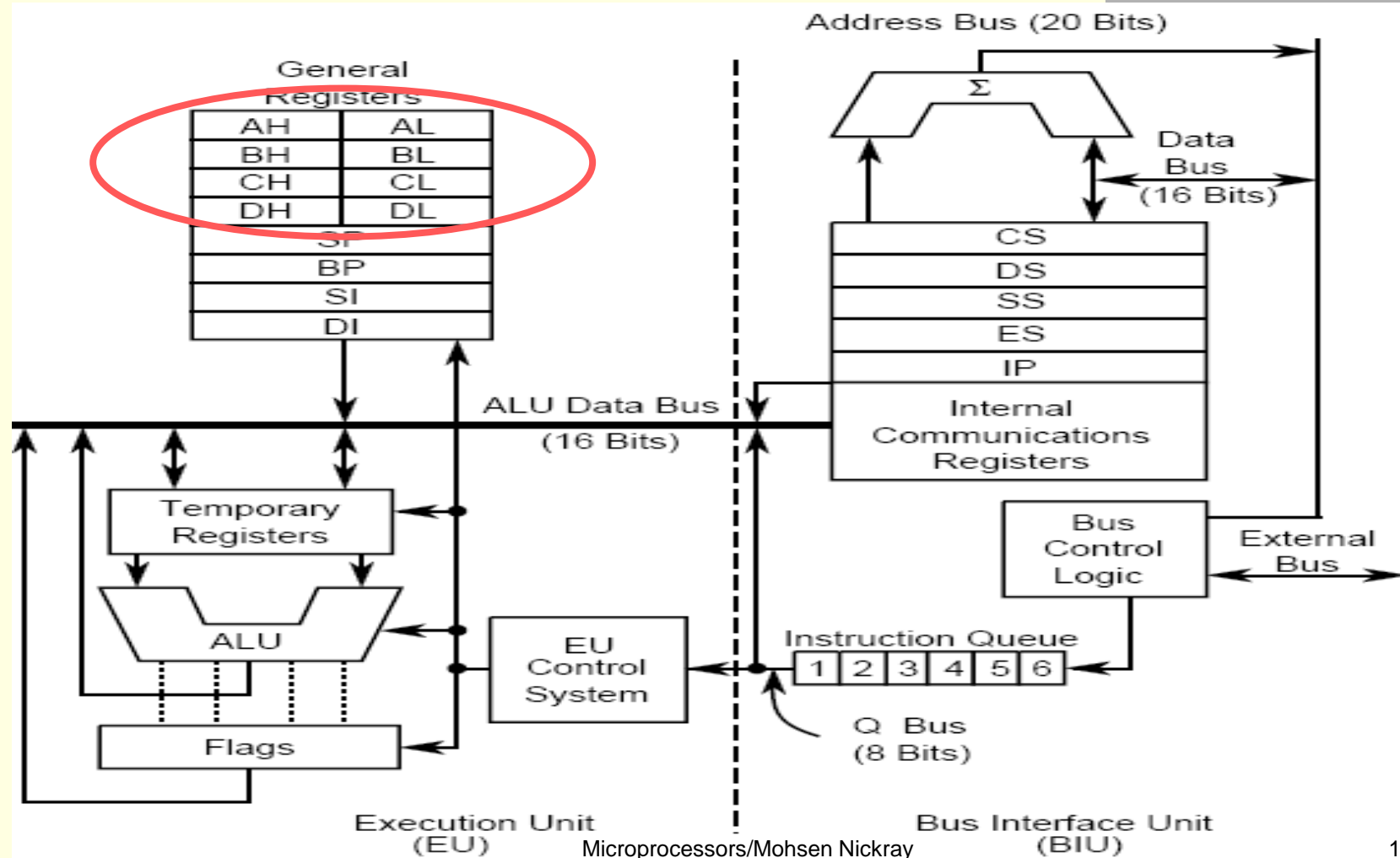
طرح معماری ۸۰۸۶



EU – واحد کنترل

- واحد کنترل مسئول هماهنگی تمام واحدهای دیگر ریزپردازنده است.
- **ALU** عملیات محاسباتی و منطقی مختلفی را روی داده ها انجام میدهد.
- دیکدر دستورات، دستورات واکشی شده توسط واحد **BIU** را ترجمه کرده و تبدیل به یکسری ریزعملهای میکند که باید توسط **EU** انجام شوند.

طرح معماری ۸۰۸۶

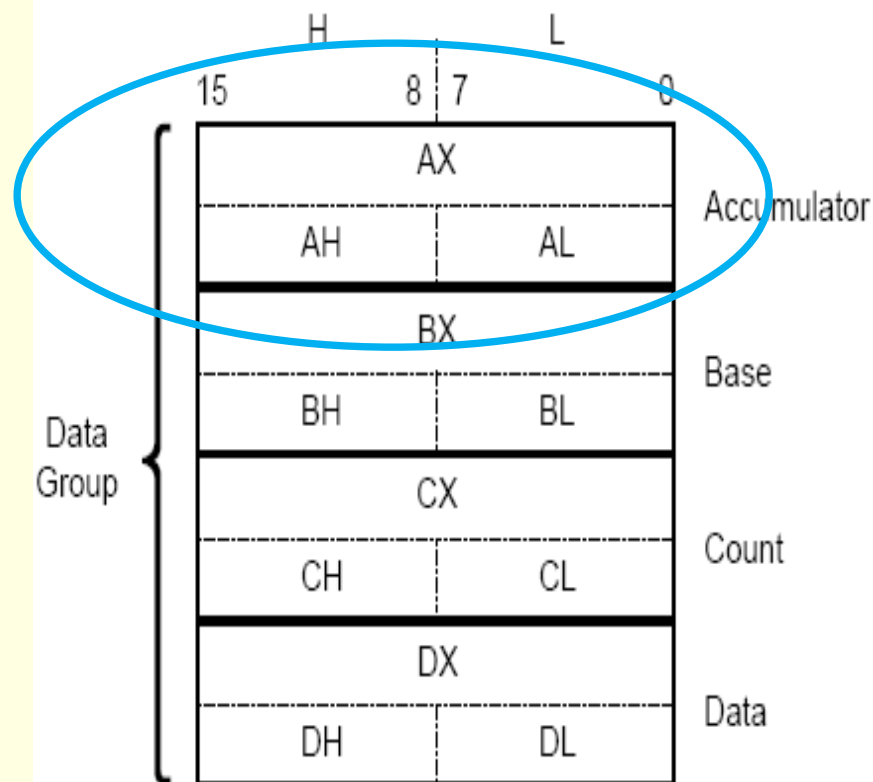


EU - رجیسترها

■ رجیسترهای عمومی برای ذخیره موقت داده ها و دستکاری داده ها (رشته ها) استفاده میشوند.

■ رجیستر Accumulator (یا AX) شامل دو تا رجیستر ۸ بیتی AL و AH است. که این دو با هم ترکیب میشوند و رجیستر ۱۶ بیتی AX را تشکیل میدهند.

■ رجیستر Accumulator میتواند برای عملیات ورودی/خروجی و دستکاری رشته ها مورد استفاده قرار گیرد.



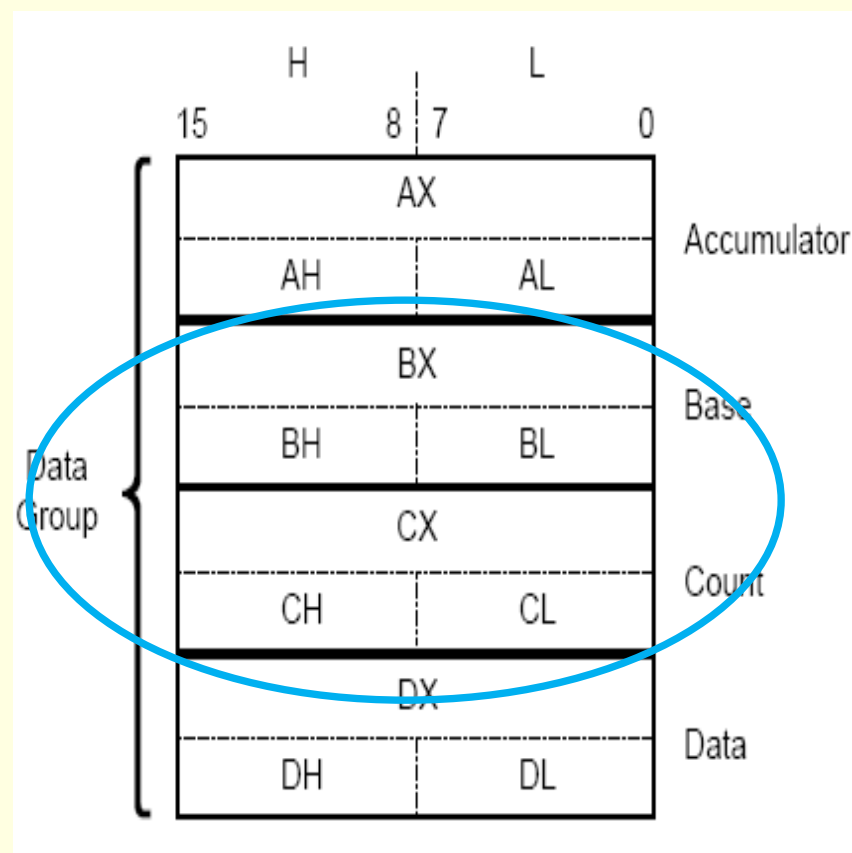
EU - رجیسترها

■ رجیستر **Base** شامل دو رجیستر ۸ بیتی **BL** و **BH** بوده که با هم ترکیب شده و رجیستر ۱۶ بیتی **BX** را تشکیل میدهند.

■ رجیستر **BX** معمولاً شامل یک اشاره گر داده است که برای **Base indexed** (آدرس خانه ابتدای رشته ها و آرایه ها) و آدرس دهی غیرمستقیم استفاده میشود.

■ رجیستر **Count** شامل دو رجیستر ۸ بیتی **CL** و **CH** بوده که میتواند با هم ترکیب شده و تشکیل یک رجیستر ۱۶ بیتی **CX** را بدهند.

■ رجیستر **Count** میتواند بعنوان یک شمارنده در کار با آرایه ها و رشته ها و بعنوان یکی از عملوندها در دستورات شیفت و **rotate** استفاده شود و یا تعداد گردش یک لوپ را نشان دهد.

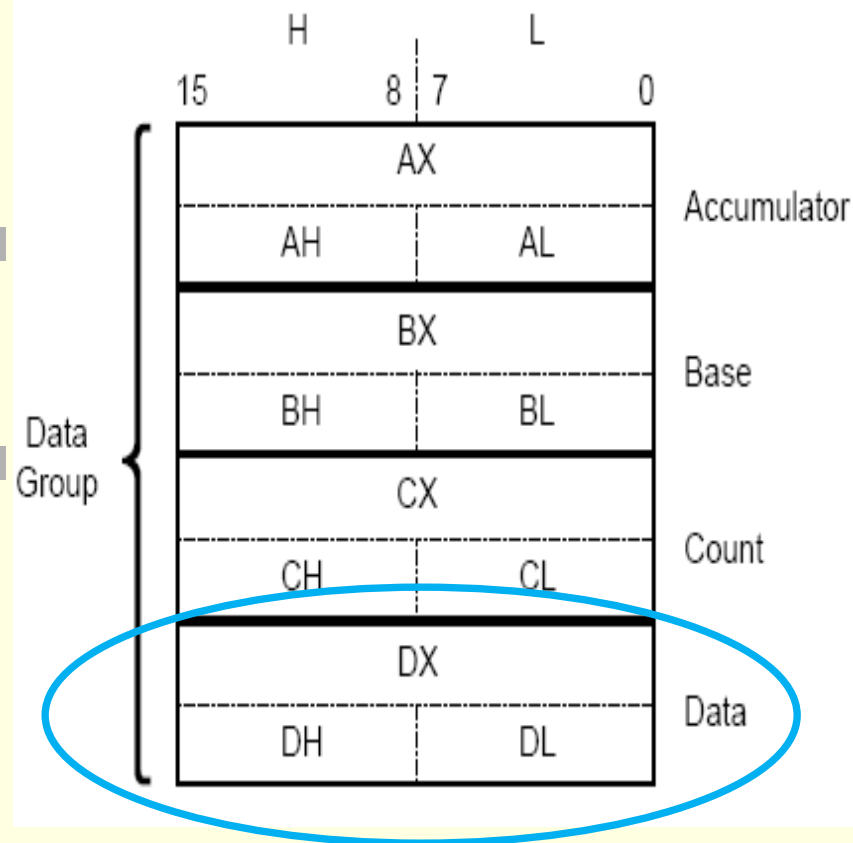


EU - رجیسترها

■ رجیستر Data شامل دو رجیستر ۸ بیتی DL و DH بوده که میتوانند ترکیب شده و رجیستر ۱۶ بیتی DX را تشکیل دهند.

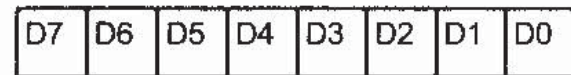
■ رجیستر Data میتواند بعنوان یک شماره پورت ورودی/خروجی مورد استفاده قرار گیرد.

■ در دستورهایی ضرب و تقسیم صحیح ۳۲ بیتی رجیستر DX شامل ۲ بایت باارزش عدد اولیه یا نتیجه محاسبه است.

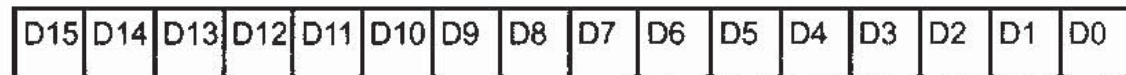


Descending order for bits

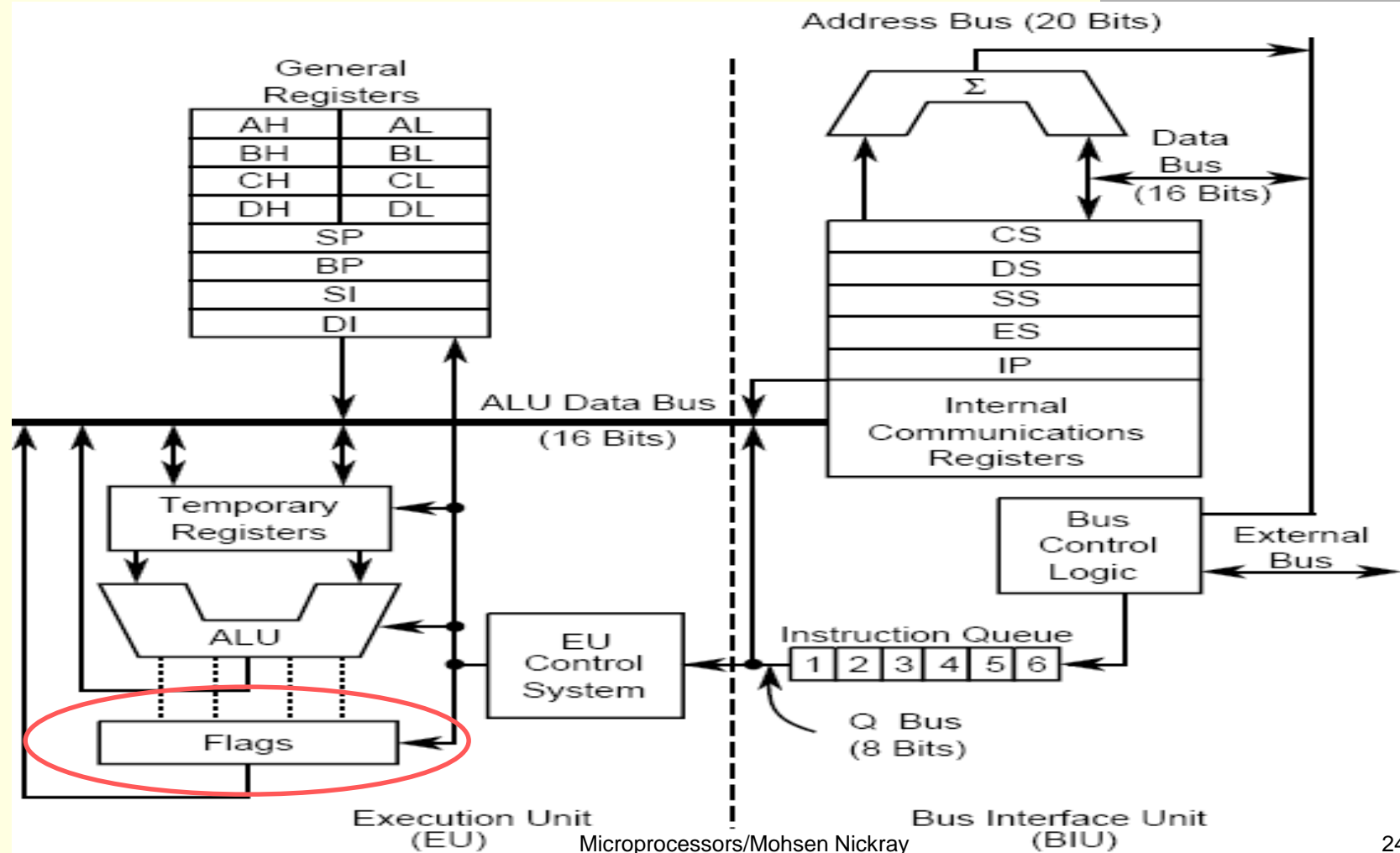
8-bit register:



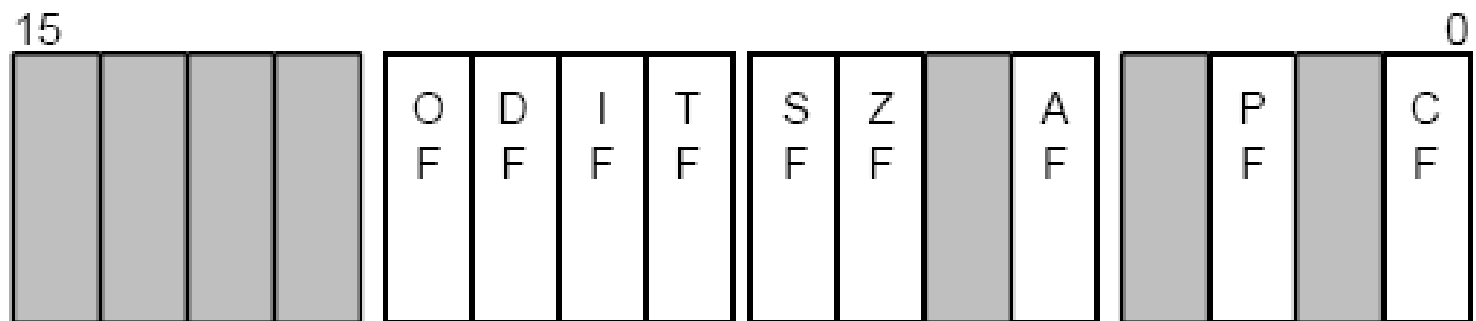
16-bit register:



طرح معماری ۸۰۸۶



Register Name:	Processor Status Word
Register Mnemonic:	PSW (FLAGS)
Register Function:	Posts CPU status information.



EU – فلگ ها

■ **Overflow Flag (OF):** اگر نتیجه محاسبه یک عدد مثبت بزرگ یا یک عدد منفی کوچک شود که در عملوند مقصد جا نشود، این فلگ یک میشود.

■ **Direction Flag (DF):** اگر یک باشد، آنگاه در دستورات کار با رشته‌ها تغییر خودکار ایندکس، بصورت کاهشی است. . اگر صفر باشد، تغییرات خودکار ایندکس، بصورت افزایشی است.

■ **Interrupt-enable Flag (IF):**

یک کردن این فلگ، maskable interrupt ها را فعال میکند.

■ **Single-step Flag (TF):** اگر یک باشد، single-step interrupt بعد از دستورالعمل بعدی اتفاق می افتد.

EU – فلگ ها

■ **Sign Flag (SF):** معادل بیت علامت نتیجه محاسبه است. اگر عدد مثبت باشد صفر، و اگر عدد منفی باشد یک میشود.

■ **Zero Flag (ZF):** اگر نتیجه محاسبه صفر شود، این فلگ یک میشود.

■ **Auxiliary carry Flag (AF):** در رجیستر AL اگر یک بیت نقلی از بیت ۳ به بیت ۴ منتقل شود، این فلگ یک میشود.

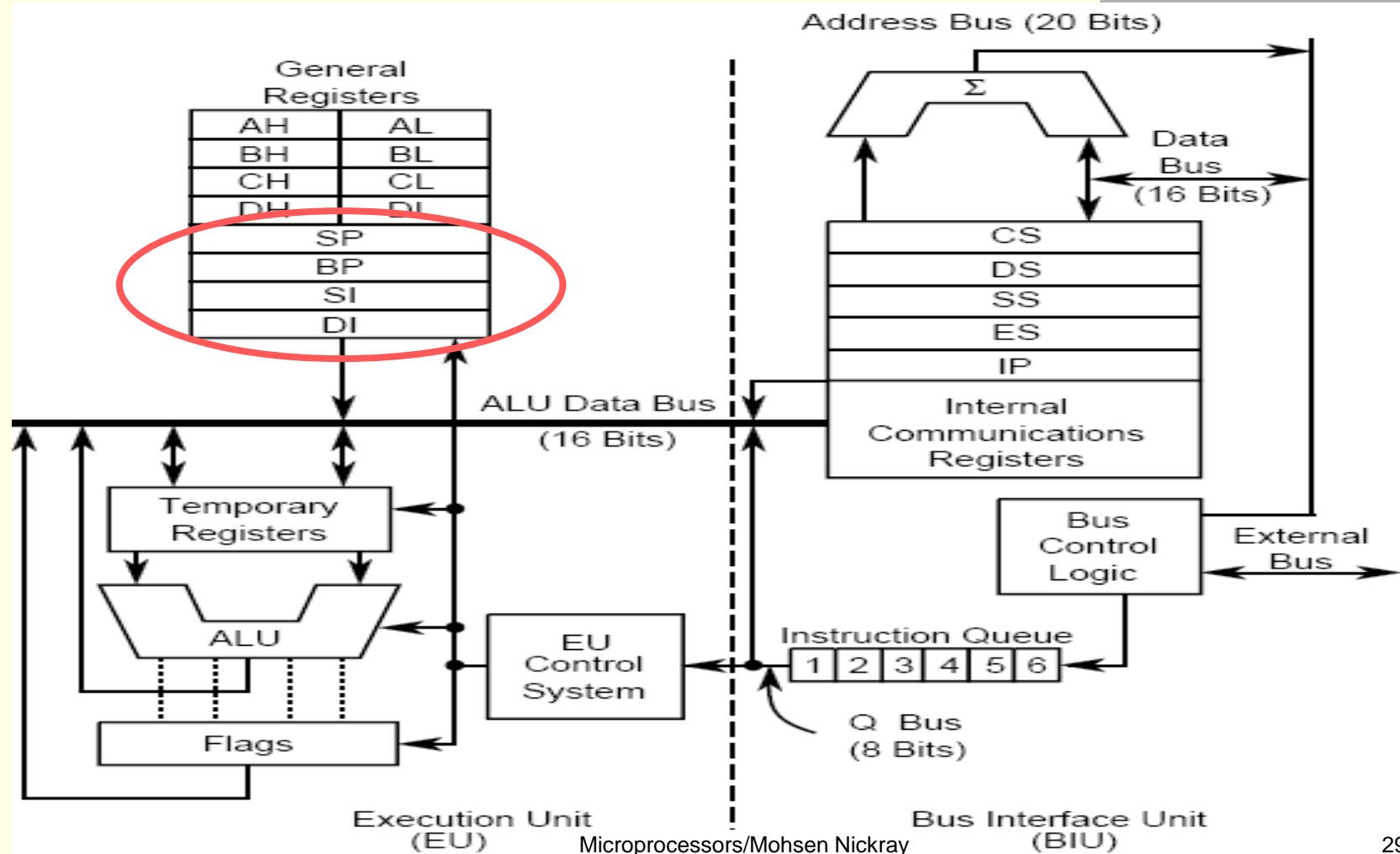
■ **Parity Flag (PF):** اگر تعداد یک های بایت کم ارزش نتیجه محاسبه زوج باشد، این فلگ یک میشود.

■ **Carry Flag (CF):** اگر نتیجه محاسبه یک بیت نقلی داشته باشد، این فلگ یک میشود.

EU – فلگ ها

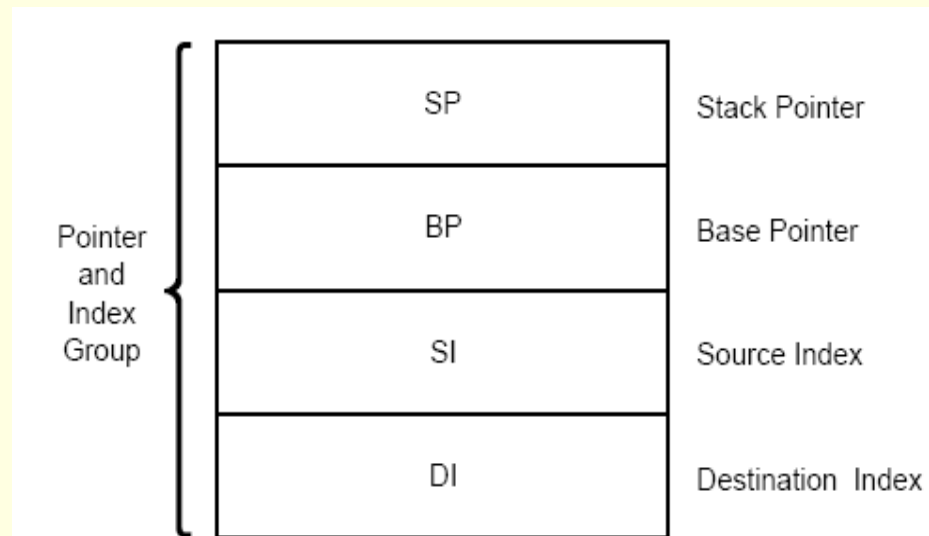
نماد	نام بیت	Reset State	عملکرد
OF	Overflow Flag	0	اگر یک شود، یک سرریز محاسباتی رخ داده است.
DF	Direction Flag	0	در دستورات رشته ها؛ اگر یک باشد، از بیت با ارزش تر به کم ارزش تر پیمایش میکند؛ و اگر صفر باشد، از بیت کم ارزش تر به بیت با ارزش تر پیمایش میکند.
IF	Interrupt Enable Flag	0	اگر یک باشد پردازنده به درخواستهای maskable interrupt رسیدگی میکند. اگر صفر باشد به آنها توجهی نمیکند.
TF	Trap Flag	0	اگر یک شود، پردازنده وارد حالت Single-Step میشود.
SF	Sign Flag	0	اگر صفر باشد، نتیجه آخرین محاسبه مثبت؛ و اگر یک باشد، منفی است.
ZF	Zero Flag	0	اگر نتیجه آخرین محاسبه صفر شود، این فلگ یک میشود.
AF	Auxiliary Flag	0	اگر یک رقم نقلی (یا قرضی) از بیت ۳ به ۴ منتقل شود، یک میشود.
PF	Parity Flag	0	اگر تعداد یک های بایت کم ارزش نتیجه آخرین محاسبه زوج باشد، یک میشود.
CF	Carry Flag	0	اگر یک رقم نقلی (یا قرضی) از بایت اول به بایت دوم منتقل شود، یک میشود.

طرح معماری ۸۰۸۶



EU – اشاره گر ها

- **Stack Pointer (SP)** : یک رجیستر ۱۶ بیتی است که به پشته اشاره میکند.
- **Base Pointer (BP)** : یک رجیستر ۱۶ بیتی است که به داده های **Segment Stack** اشاره میکند. BP معمولا برای آدرس پایه، آدرس پایه ایندکسینگ و آدرس دهی غیرمستقیم توسط رجیستر، استفاده میشود.

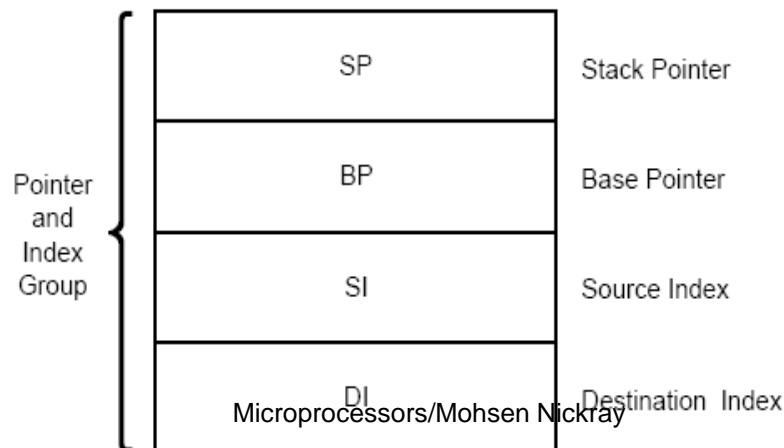


Stack Pointer (SP) is a 16-bit register pointing to program stack.

Base Pointer (BP) is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.

Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data address in string manipulation instructions.

Destination Index (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.



دستور العمل MOV

■ MOV instruction

■ MOV des, src مقدار src را به des کپی میکند. ;

■ مثال:

- MOV CL,55H
- MOV DL, CL
- MOV AH, DL

- MOV CX,EF28H
- MOV AX, CX
- MOV DI, AX
- MOV BP,DI

■ این دستور را برای رجیستر فلگ نمیتوان استفاده کرد.

■ **immediate load** را نمیتوان برای رجیسترهای سگمنت انجام داد. فقط برای رجیسترهای عمومی میتوان به کار برد.

■ اندازه مبدا و مقصد باید برابر باشد. (هر دو یک بایت یا دو بایت یا ... باشند)

Move Instruction

MOV	AX,58FCH	;move 58FCH into AX
MOV	DX,6678H	;move 6678H into DX
MOV	SI,924BH	;move 924B into SI
MOV	BP,2459H	;move 2459H into BP
MOV	DS,2341H	;move 2341H into DS
MOV	CX,8876H	;move 8876H into CX
MOV	CS,3F47H	;move 3F47H into CS
MOV	BH,99H	;move 99H into BH

MOV Instruction

Values cannot be loaded directly into any segment register (CS, DS, ES, or SS). To load a value into a segment register, first load it to a nonsegment register and then move it to the segment register, as shown next.

```
MOV  AX,2345H    ;load 2345H into AX
MOV  DS,AX        ;then load the value of AX into DS

MOV  DI,1400H     ;load 1400H into DI
MOV  ES,DI        ;then move it into ES, now ES=DI=1400
```

دستور العمل ADD

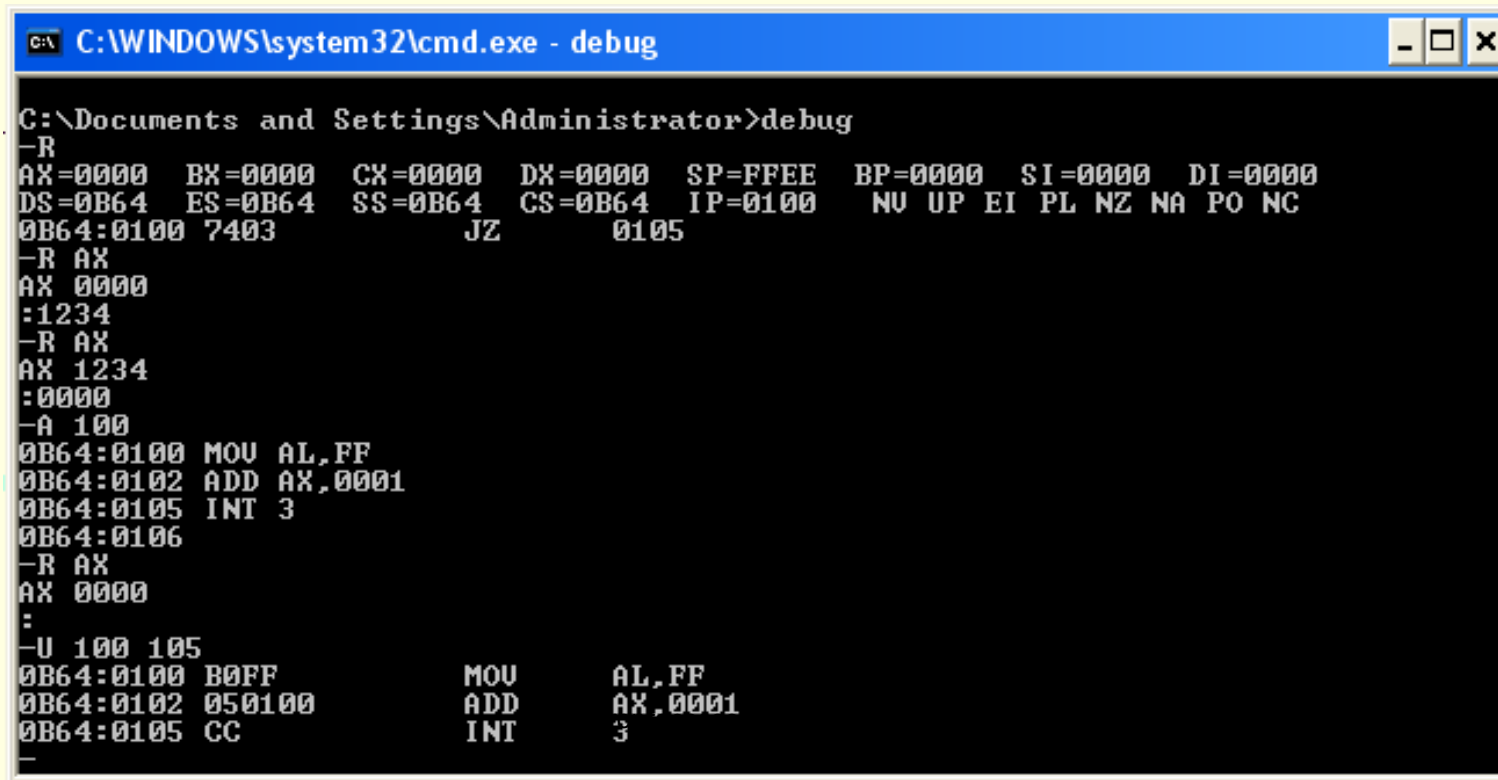
- ADD instruction
 - مقدار src را به des اضافه و با آن جمع میکند. ;
ADD des, src
 - مثال:
 - MOV AL,55H
 - MOV CL,23H
 - ADD AL,CL ;

 - MOV DH,25H
 - ADD DH,34H ; عملوند دوم از نوع immediate است.

 - MOV CX,345H
 - ADD CX,679H
- اندازه مبدا و مقصد باید برابر باشد. (هر دو یک بایت یا دو بایت یا ... باشند)

اجرا و خطایابی برنامه

- R <register name>
- A <starting address>
- U <start> <end> or U <start> <L number>



```
C:\WINDOWS\system32\cmd.exe - debug

C:\Documents and Settings\Administrator>debug
-R
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B64 ES=0B64 SS=0B64 CS=0B64 IP=0100  NU UP EI PL NZ NA PO NC
0B64:0100 7403          JZ      0105
-R AX
AX 0000
:1234
-R AX
AX 1234
:0000
-A 100
0B64:0100 MOV AL,FF
0B64:0102 ADD AX,0001
0B64:0105 INT 3
0B64:0106
-R AX
AX 0000
:
-U 100 105
0B64:0100 B0FF          MOV     AL,FF
0B64:0102 050100      ADD     AX,0001
0B64:0105 CC          INT     3
-
```

اجرا و خطایابی برنامه (ادامه)

- $G < = \text{starting address} > < \text{stop address(es)} >$

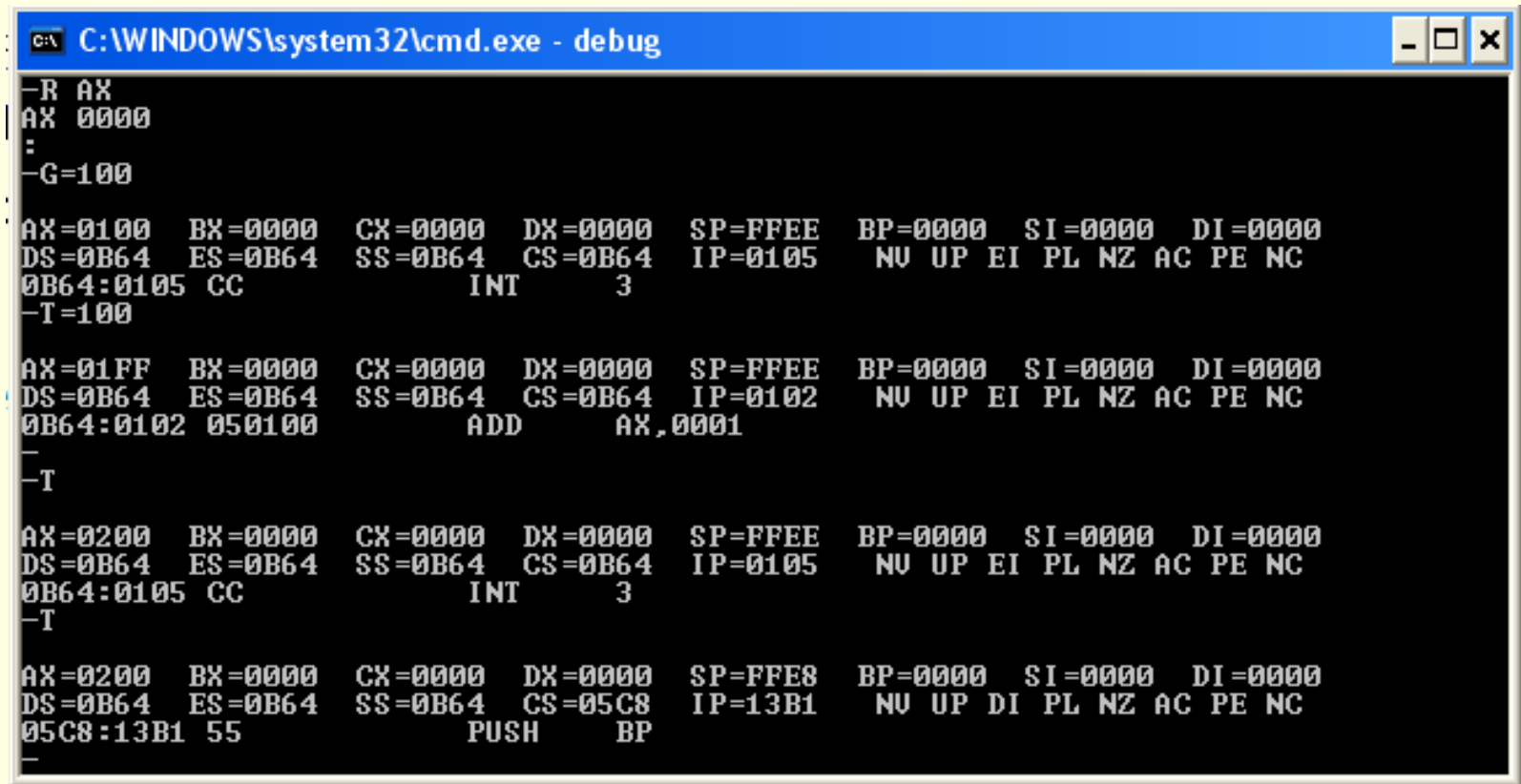
```
C:\WINDOWS\system32\cmd.exe - debug
0B64:0100 MOV AL,FF
0B64:0102 ADD AX,0001
0B64:0105 INT 3
0B64:0106
-R AX
AX 0000
:
-U 100 105
0B64:0100 B0FF          MOV     AL,FF
0B64:0102 050100        ADD     AX,0001
0B64:0105 CC           INT     3
-G 0100

AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B64  ES=0B64  SS=0B64  CS=0B64  IP=0100  NU UP EI PL NZ NA PO NC
0B64:0100 B0FF          MOV     AL,FF
-R AX
AX 0000
:
-G=100

AX=0100  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B64  ES=0B64  SS=0B64  CS=0B64  IP=0105  NU UP EI PL NZ AC PE NC
0B64:0105 CC           INT     3
-
```

اجرا و خطایابی برنامه (ادامه)

- T < = starting address> <number>



```
C:\WINDOWS\system32\cmd.exe - debug
-R AX
AX 0000
:
-G=100
AX=0100 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B64 ES=0B64 SS=0B64 CS=0B64 IP=0105  NU UP EI PL NZ AC PE NC
0B64:0105 CC          INT     3
-T=100
AX=01FF BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B64 ES=0B64 SS=0B64 CS=0B64 IP=0102  NU UP EI PL NZ AC PE NC
0B64:0102 050100      ADD     AX,0001
-
-T
AX=0200 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B64 ES=0B64 SS=0B64 CS=0B64 IP=0105  NU UP EI PL NZ AC PE NC
0B64:0105 CC          INT     3
-T
AX=0200 BX=0000 CX=0000 DX=0000 SP=FFE8 BP=0000 SI=0000 DI=0000
DS=0B64 ES=0B64 SS=0B64 CS=05C8 IP=13B1  NU UP DI PL NZ AC PE NC
05C8:13B1 55          PUSH    BP
-
```

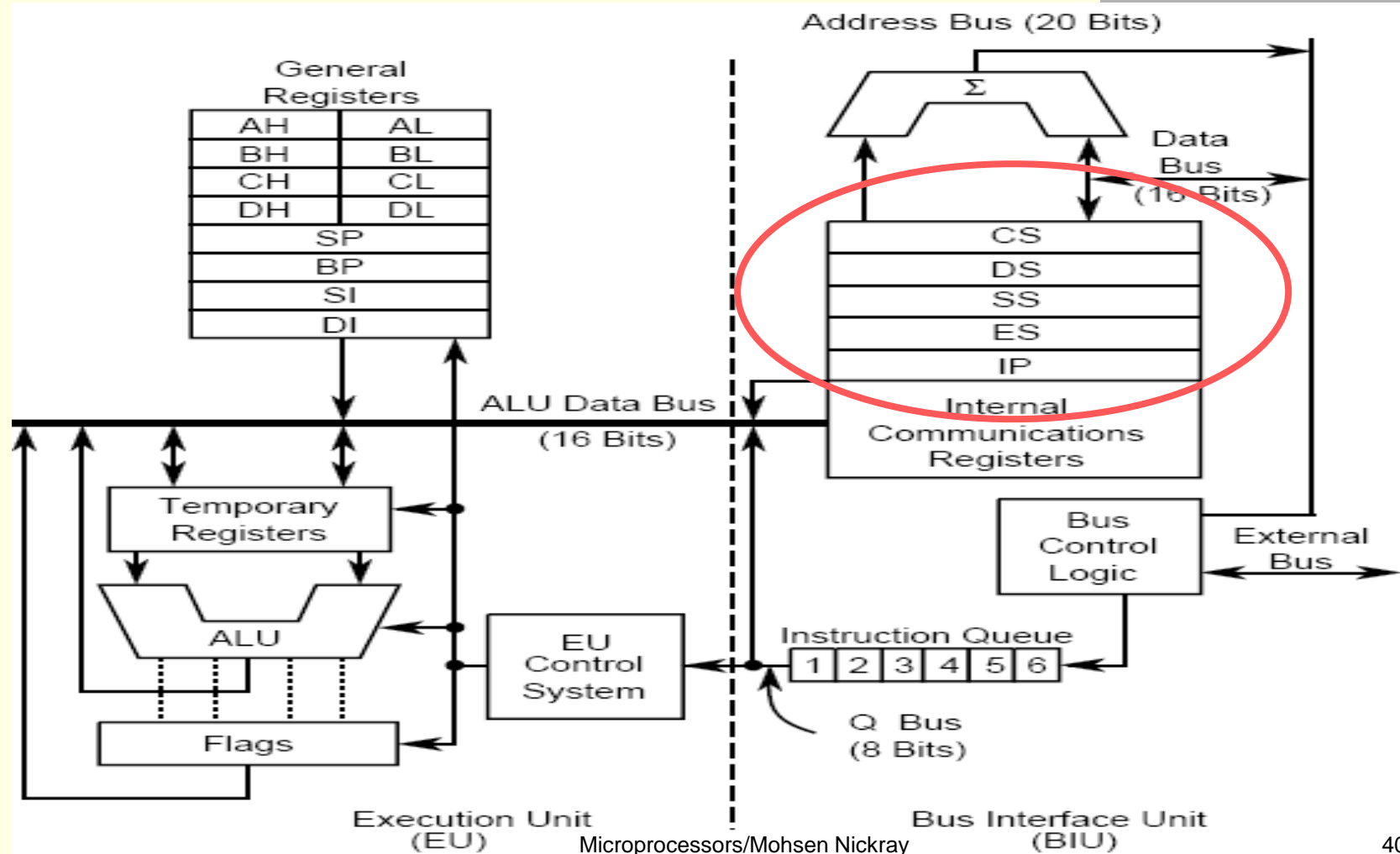
اجرا و خطایابی برنامه (ادامه)

- F <s> <e> <data> or F <s> <L n> <data>
- E <address> <data list>
- D <s> <e> or D <s> <L n>

```
C:\WINDOWS\system32\cmd.exe - debug
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>debug
-F 0100 0110 FF
-D 0100 0120
0B64:0100  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  .....
0B64:0110  FF 0B C0 75 18 50 53 52-57 2E C7 06 34 00 53 0B  ....u.PSRW...4.S.
0B64:0120  B4  .....
-E 0100 01 02 03 04 05 06 07
-D 0100 0120
0B64:0100  01 02 03 04 05 06 07 FF-FF FF FF FF FF FF FF FF FF  .....
0B64:0110  FF 0B C0 75 18 50 53 52-57 2E C7 06 34 00 53 0B  ....u.PSRW...4.S.
0B64:0120  B4  .....
-
```

طرح معماری ۸۰۸۶



رجیسترهای Segment

- حافظه پردازنده ۸۰۸۶ به چهار قسمت تقسیم شده است:
 - Code segment (program memory)
 - Data segment (data memory)
 - Stack memory (stack segment)
 - Extra memory (extra segment)

نمادگذاری Segment:Offset

- کل مقدار قابل آدرس دهی حافظه 1MB است.
- بیشتر دستورالعملهای پردازنده از یک اشاره گر ۱۶ بیتی استفاده میکنند که میتواند بصورت مؤثر 64KB از حافظه را آدرس دهی کند.
- برای دسترسی به بیرون این 64KB، پردازنده از رجیسترهای سگمنت مخصوصی استفاده میکند که محل دقیق کد، دیتا یا پشته را در این 1MB تعیین میکند.

نمادگذاری Segment:Offset

در حافظه باید یک طرح ساده و منسجم وجود داشته باشد که بر اساس آن بتوان داده های ۰ و ۱ که بصورت سریال و پشت سرهم قرار دارند را از ابتدا تا انتهای حافظه سازماندهی کرد.

در پردازنده ۸۰۸۶ این طرح قطعه بندی یا Segmentation نام دارد.

هر آدرس دو قسمت دارد: یک قسمت segment و یک قسمت offset. (segment:offset)

قسمت Segment برابر است با آدرس شروع یک قسمت از این 64KB جزء (تکه) حافظه ضرب در عدد ۱۶.

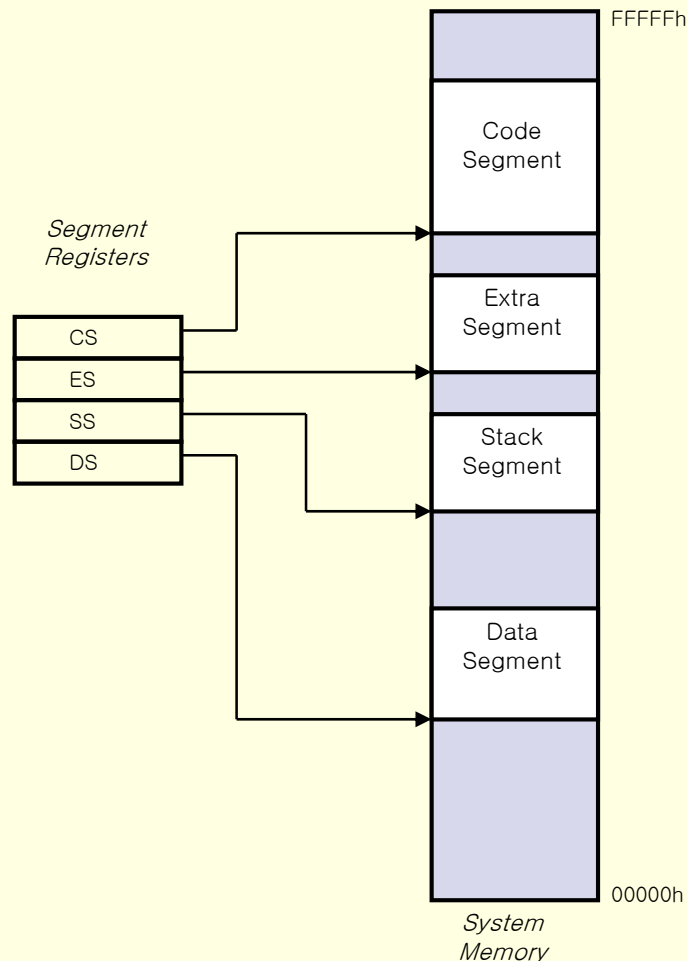
قسمت offset برابر است با مکانی در داخل این قسمت مشخص شده توسط segment (مکانی در داخل یکی از این 64KB تکه ها)

پس آدرس مطلق عبارت است:

$$\text{Absolute address} = (\text{segment} * 16) + \text{offset}$$

حافظه سگمنت شده (Segmented Memory)

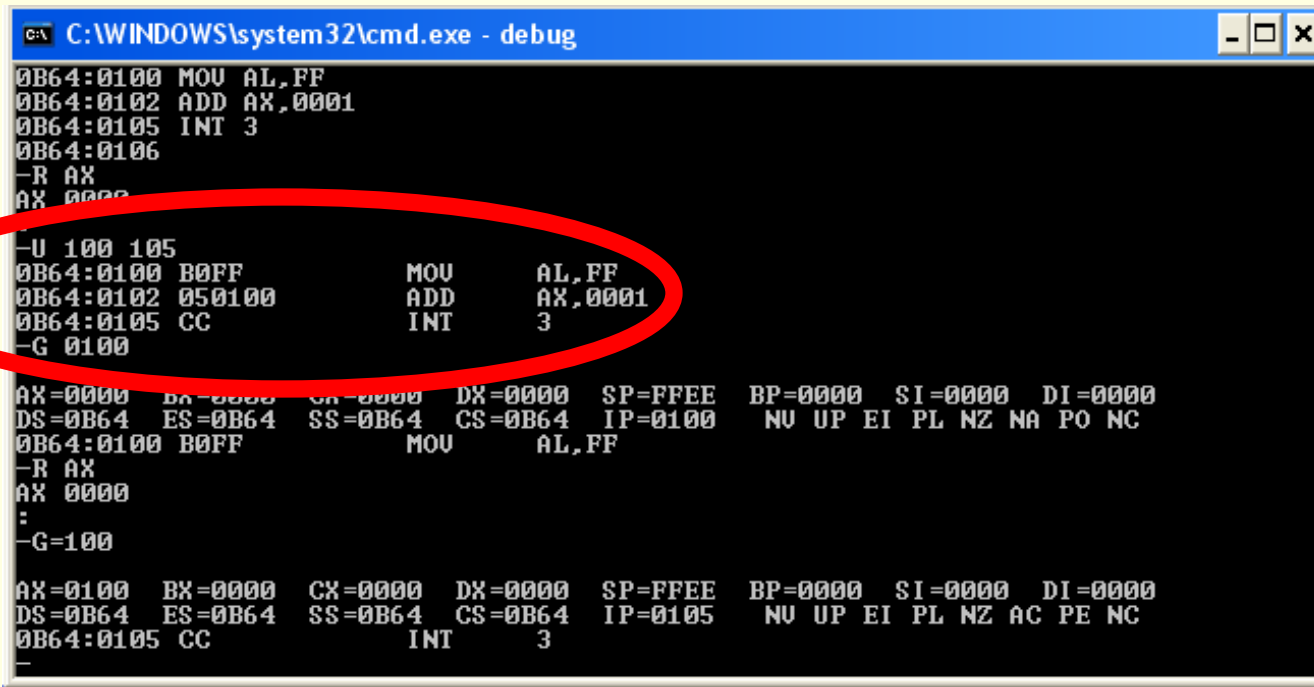
یک مثال:



- Logical, Segmented Address:
0FE6:012Bh
- Offset, Index Address:
012Bh
- Physical Address:
0FE60h → 65120
+ 012Bh → 299
0FF8Bh → 65149

سگمنت کد (Code Segment)

- برای اجرای یک برنامه، پردازنده ۸۰۸۶ دستورات را از code segment واکنشی میکند.
- مقدار آدرس منطقی دستورالعمل همیشه CS:IP است.



```
C:\WINDOWS\system32\cmd.exe - debug
0B64:0100 MOV AL,FF
0B64:0102 ADD AX,0001
0B64:0105 INT 3
0B64:0106
-R AX
AX 0000
-U 100 105
0B64:0100 B0FF      MOV     AL,FF
0B64:0102 050100     ADD     AX,0001
0B64:0105 CC        INT     3
-G 0100
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B64  ES=0B64  SS=0B64  CS=0B64  IP=0100  NU UP EI PL NZ NA PO NC
0B64:0100 B0FF      MOV     AL,FF
-R AX
AX 0000
:
-G=100
AX=0100  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B64  ES=0B64  SS=0B64  CS=0B64  IP=0105  NU UP EI PL NZ AC PE NC
0B64:0105 CC        INT     3
-
```

مثال

If CS = 24F6H and IP = 634AH, show:

- (a) The logical address
 - (b) The offset address
- and calculate:
- (c) The physical address
 - (d) The lower range
 - (e) The upper range of the code segment

Solution:

سگمنت داده (Data Segment)

■ در ۸۰۸۶، محلی از حافظه را که داده ها در آن ذخیره میشود را Data Segment (DS) میگویند.

■ DataSegment بوسیله یک رجیستر ۱۶ بیتی که شامل آدرس 64KB قسمت، است قابل دسترسی میباشد.

DS:0200 = 25

DS:0201 = 12

DS:0202 = 15

DS:0203 = 1F

DS:0204 = 05

```
MOV     AL,0           ;clear AL
ADD     AL,[0200]       ;add the contents of DS:200 to AL
ADD     AL,[0201]       ;add the contents of DS:201 to AL
ADD     AL,[0202]       ;add the contents of DS:202 to AL
ADD     AL,[0203]       ;add the contents of DS:203 to AL
ADD     AL,[0204]       ;add the contents of DS:204 to AL
```

سگمنت داده (Data Segment)

■ در ۸۰۸۶ فقط رجیسترهای BX و SI و DI میتوانند بعنوان نگهدارنده آدرس offset استفاده شوند.

MOV	AL,0	;initialize AL
MOV	BX,0200H	;BX points to the offset addr of first byte
ADD	AL,[BX]	;add the first byte to AL
INC	BX	;increment BX to point to the next byte
ADD	AL,[BX]	;add the next byte to AL
INC	BX	;increment the pointer
ADD	AL,[BX]	;add the next byte to AL
INC	BX	;increment the pointer
ADD	AL,[BX]	;add the last byte to AL

سگمنت داده (Data Segment)

■ هنگامی که از یک داده ۱۶ بیتی استفاده میکنیم، چه اتفاقی می افتد؟

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

■ پردازنده ۸۰۸۶ یک پردازنده little endian است.

DS:1500 = F3

DS:1501 = 35

Little Endian / Big Endian

for the 68000:

MOVE.W	#513, D0	;	مقدار ۵۱۳ را به ۱۶ بیت پایین D0 منتقل میکند.
MOVE.W	D0,4	;	دوبایت پایین D0 را در حافظه با آدرس ۴ ذخیره میکند.

for the 80x86:

MOV	AX,513	;	مقدار ۵۱۳ را در AX (۱۶ بیت) قرار میدهد.
MOV	[4],AX	;	مقدار AX را در حافظه با آدرس ۴ ذخیره میکند.

680x0 (big-endian)

address	8-bits wide
0	
1	
2	
3	
4	02
5	01
6	
7	
8	
9	
A	
.	
.	
.	
.	

80x86 (small endian)

address	8-bits wide
0	
1	
2	
3	
4	01
5	02
6	
7	
8	
9	
A	
.	
.	
.	
.	

Extra Segment

- در ۸۰۸۶، در برنامه های عادی معمولاً استفاده نمیشود.
- استفاده از آن برای عملیات های کار با رشته ها لازم است.
- رجیستر **Extra Segment (ES)** یک رجیستر ۱۶ بیتی است که شامل آدرس 64KB قسمت، است که معمولاً در دیتا برنامه است.
- بصورت پیشفرض پردازنده، فرض میکند که رجیستر **DI** برای ارجاع به **ES** رجوع میکند (در دستورالعملهای کار با رشته ها و دستکاری آنها).

سگمنت پشته (Stack Segment)

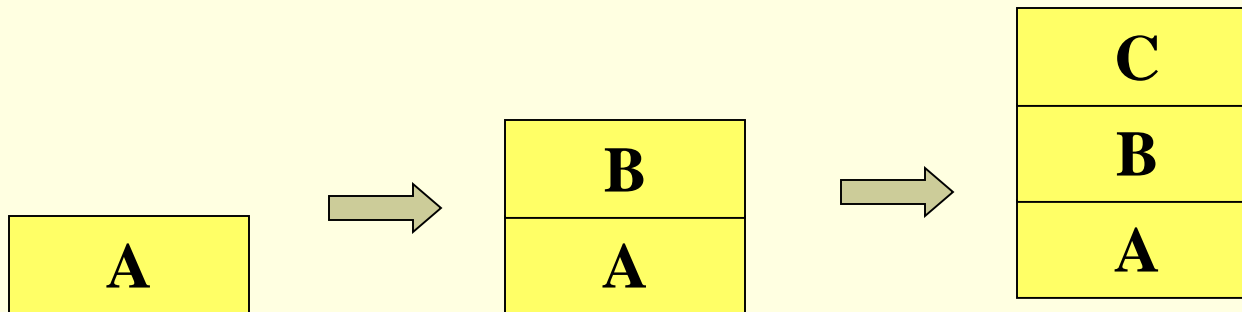
■ رجیستر Stack Segment (SS) یک رجیستر ۱۶ بیتی بوده که شامل آدرس 64KB قسمت، در قسمت پشته برنامه.

■ بصورت پیشفرض، پردازنده فرض میکند که همه داده هایی که ارجاع داده میشوند توسط رجیستر اشاره گر پشته (SP) و رجیستر اشاره گر پایه (BP)، در StackSegment قرار دارند.

ذخیره داده توسط پشته

کلمه «پشته» به این دلیل استفاده میشود چون ذخیره و بازیابی کلمات در محدوده ای از حافظه که پشته است، همانند دسترسی به کلمات یک پشته است.

یک پشته از خانه هایی را مطابق زیر تصور کنید. برای اینکه یک پشته ایجاد کنیم باید ابتدا A را قرار دهیم، سپس B و بعد از آن C را روی آنها قرار دهیم.



توجه کنید که شما فقط به آخرین عنصری که در پشته قرار داده اید (بالای پشته - TOS) دسترسی دارید. یعنی برای بازیابی عناصر، بازیابی دارای ترتیب برعکس نسبت به پر شدن پشته است. یعنی به ترتیب C و B و A بازیابی میشوند.

پشته

- یکی از کاربردهای متداول پشته معمولا در ذخیره داده های موقتی استفاده میشود.
- پشته توسط `SS:SP (StackSegment:StackPointer)` که ترکیب `segment` و `offset` است، قابل دسترسی است.
- برخی از دستورالعملها که در کار با پشته استفاده میشوند عبارت اند از:
`push , pop , call , ret , many , ...`
- اگر شما نیاز به ذخیره داده های موقت داشتید، قطعا پشته بهترین مکان برای این کار است.

ذخیره داده در پشته x86 توسط push

■ رجیستر SP (Stack Pointer) برای دسترسی به عناصر موجود در پشته استفاده میشود. که این اشاره گر به آخرین عنصر پشته که در آن قرار داده شده، اشاره میکند.

■ عمل push یک مقدار را در پشته ذخیره میکند:

■ `PUSH AX ; SP= SP-2, M[SP] ← AX`

■ که این دستورالعمل در واقع با دو دستورالعمل زیر که پشت سرهم هستند معادل است:

■ `sub SP, 2` ; دو واحد کم میکند. SP از اشاره گر
■ `mov [SP], AX` ; مقدار را در پشته قرار میدهد.

■ پشته فقط توسط عملوندهای ۱۶ و ۳۲ بیتی قابل دسترسی است.

شبه سازی عمل push

PUSH AX قبل از دستور

PUSH Ax بعد از دستور

high memory

lastval
????
????
????
????
????
????
????
????

← SP

در این تصاویر حافظه
بصورت کلمات ۱۶ بیتی
نشان داده شده است. با
توجه به اینکه عملیات
پشته همیشه ۱۶ یا ۳۲
بیتی هستند.

Low memory

high memory

lastval
ahal
????
????
????
????
????
????
????

← SP

(new SP =
old SP-2)

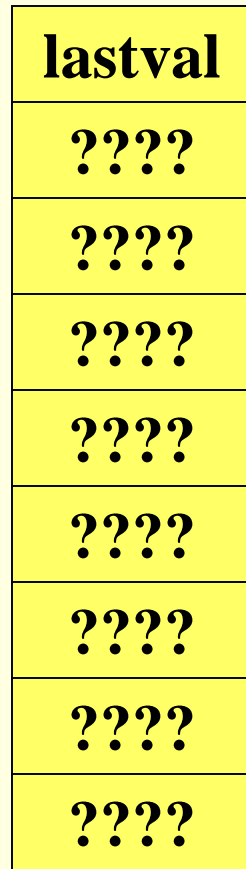
Low memory

چند push پشت سرهم

قبل از دستورات

بعد از اجرای دستورات

high memory

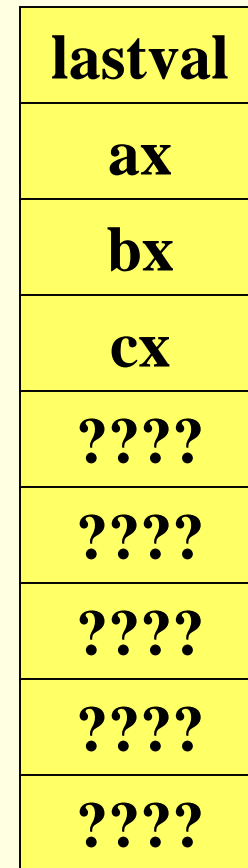


← SP

PUSH AX
PUSH BX
PUSH CX

Low memory

high memory



← SP

Low memory

خواندن داده از پشته x86 توسط pop

دستور pop یک مقدار را از پشته بازیابی میکند:

POP AX ; $AX \leftarrow M[SP]$, $SP = SP + 2$

دستور بالا با این دو دستور پشت سرهم معادل است:

mov AX, [SP] ; مقداری را از بالای پشته میخواند.

add sp, 2 ; اشاره گر پشته را دو واحد زیاد میکند.

شبه سازی عمل push

قبل از دستور POP Ax

بعد از دستور POP Ax

high memory

FF65
23AB
????
????
????
????
????
????
????

← SP

در این تصاویر حافظه بصورت کلمات ۱۶ بیتی نشان داده شده است. با توجه به اینکه عملیات پشته همیشه ۱۶ یا ۳۲ بیتی هستند.

high memory

FF65
23AB
????
????
????
????
????
????
????

← SP

AX = 23AB

low memory

low memory

چند pop پشت سرهم

قبل از اجرا

بعد از اجرای دستورات POP

high memory

FF65
23AB
357F
D21B
38AC
23F4
????
????
????

pop AX
pop BX
pop CX

← SP

low memory

high memory

FF65
23AB
357F
D21B
38AC
23F4
????
????
????

← SP

AX = 38AC
BX = D21B
CX = 357F

low memory

Stack Overflow, Underflow

■ اگر شما داده ها را روی پشته **push** کنید ولی بعدا آنها را **pop** نکنید و اینکار را همینطور ادامه دهید، آنگاه پشته بیشتر از فضای اختصاص داده شده به آن رشد میکند. در این حالت با انجام عمل **push** در جایی که پشته نیست داده ها را مینویسید. (مثلا کد، دیتا ، ...)

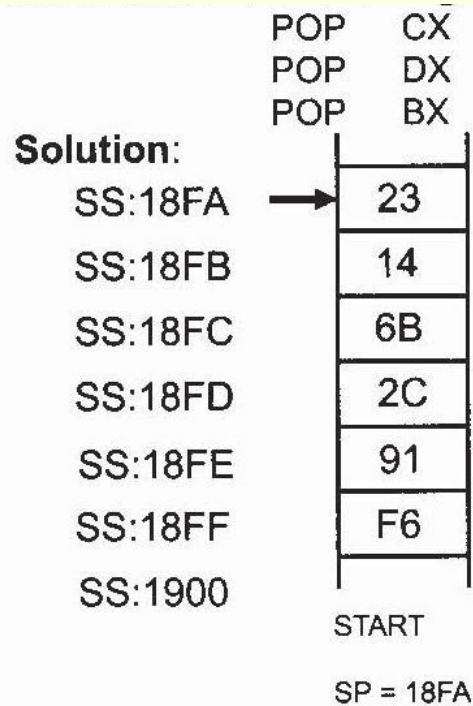
■ به این حالت **Stack Overflow** میگویند.

■ همچنین چنانچه داده ها را بیش از اندازه ای که آنها را **push** کرده ایم، **pop** کنیم، آنگاه اشاره گر پشته بیش از محل آغاز کاهش می یابد. که باز هم از محل اختصاص داده به پشته خارج می شود. به این حالت **Stack Underflow** میگویند.

■ پس: ما باید به اندازه ای که مورد نیاز داریم به پشته فضا از حافظه اختصاص دهیم. هم چنین در هنگام کار با پشته به همان اندازه که **push** میکنیم، به همان اندازه **pop** انجام دهیم.

Example

فرض کنید شکل زیر پشته است. و $SP=18FA$. محتوای پشته و رجیسترهای زیر را بعد اجرای هر دستور زیر مشخص کنید؟



حالت های آدرس دهی

انواع عملوندها در یک دستورالعمل:

■ رجیستر - مثل AX

■ حالت Immediate address - مثل 12H

■ آدرس دهی حافظه:

■ Direct addressing - مثل [3965]

■ Register indirect - مثل [BX]

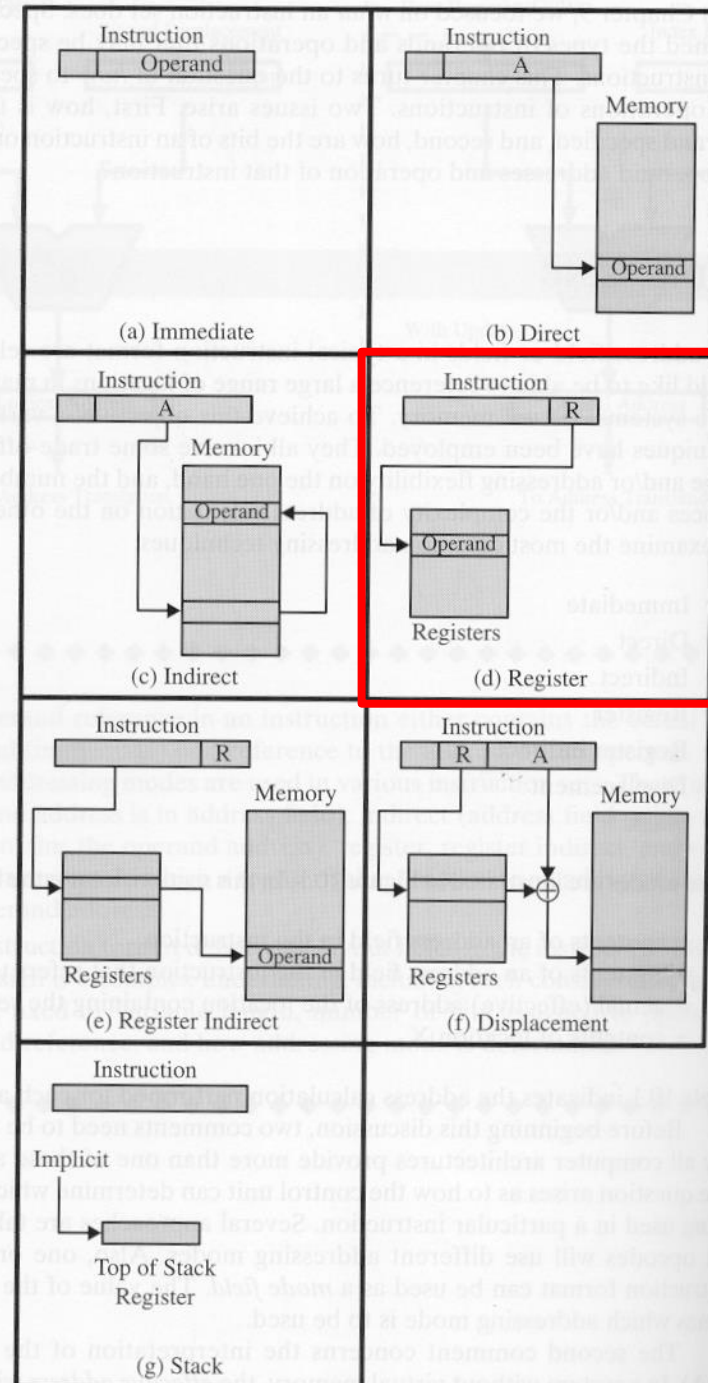
■ Based relative addressing

مثل: [BX+6], [BP]-10

■ Indexed relative addressing

مثل: [SI+5], [DI]-8

■ Based indexed addressing



Register addressing mode

- Register addressing mode involves the use of registers to hold data
- Example:

```
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
```

MOV AL,CX !!!Error

حالت های آدرس دهی

انواع عملوندها در یک دستورالعمل:

■ رجیستر - مثل AX

■ حالت Immediate address - مثل 12H

■ آدرس دهی حافظه:

■ Direct addressing - مثل [3965]

■ Register indirect - مثل [BX]

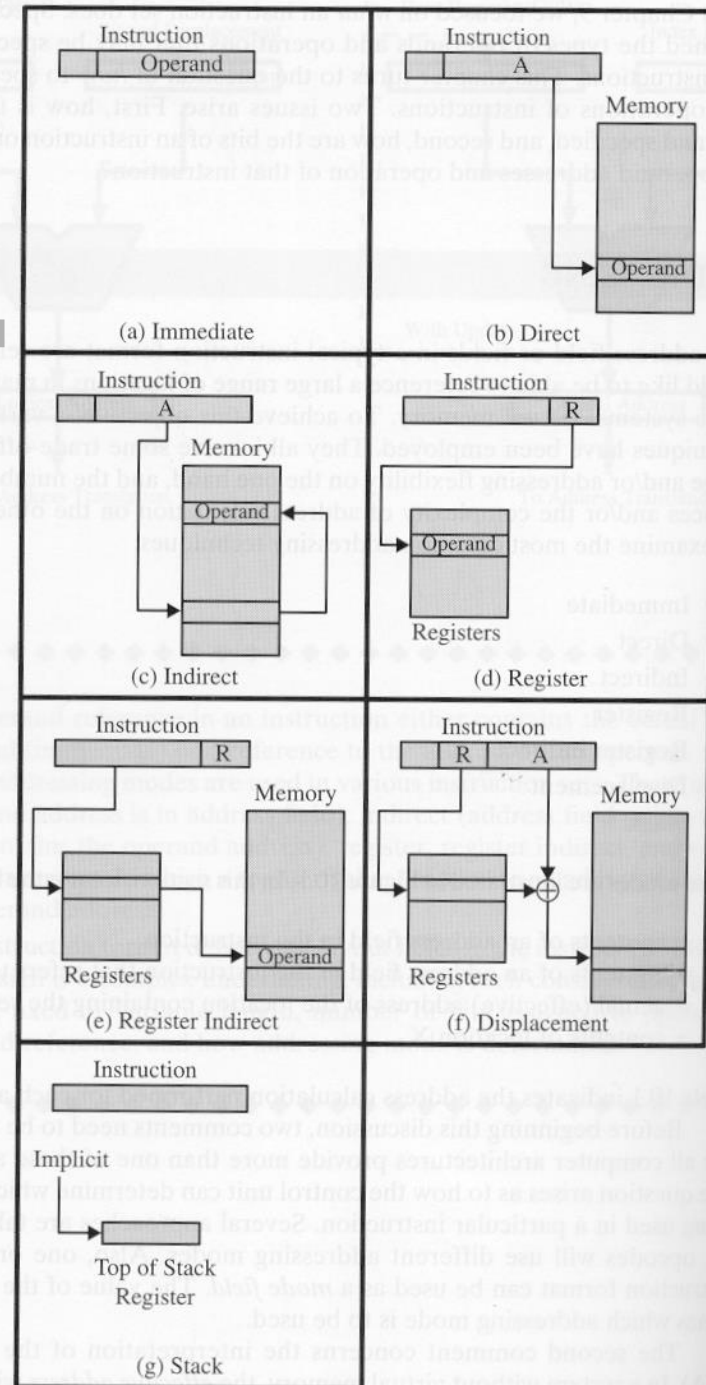
■ Based relative addressing

مثل: [BX+6], [BP]-10

■ Indexed relative addressing

مثل: [SI+5], [DI]-8

■ Based indexed addressing



Immediate addressing mode

- The source operand is a constant
- As the name implies, when the instruction is assembled, the operands come immediately after the operand
- Examples:

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

- MOV DS,0123H **!! Error** ==> MOV AX,0123H
MOV DS,AX

حالت های آدرس دهی

انواع عملوندها در یک دستورالعمل:

■ رجیستر - مثل AX

■ حالت Immediate address - مثل 12H

■ آدرس دهی حافظه:

■ Direct addressing - مثل [3965]

■ Register indirect - مثل [BX]

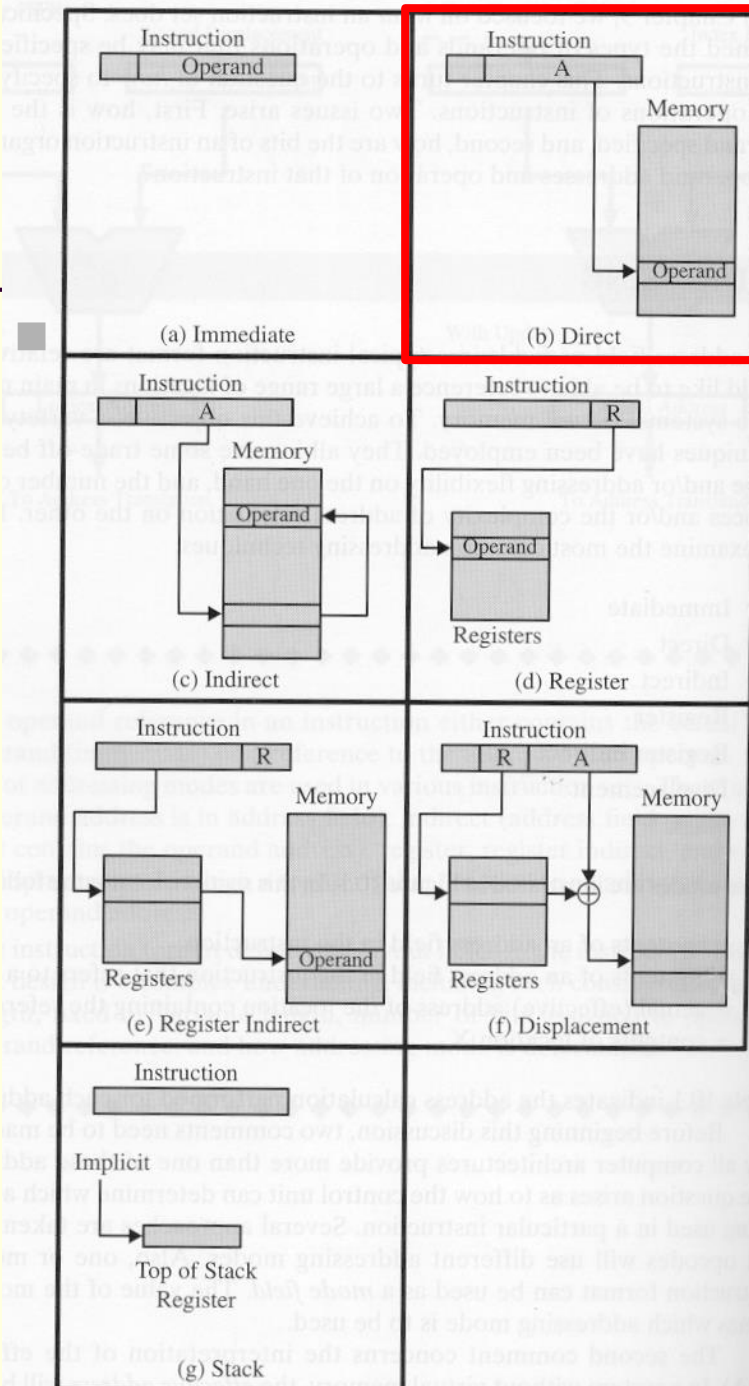
■ Based relative addressing

مثل: [BX+6], [BP]-10

■ Indexed relative addressing

مثل: [SI+5], [DI]-8

■ Based indexed addressing



آدرس دهی مستقیم (Direct addressing)

- داده در بعضی از مکانهای حافظه قرار دارد.
- آدرس داده بصورت فوری بعد از دستورالعمل می آید.
- مثال:

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

حالت های آدرس دهی

انواع عملوندها در یک دستورالعمل:

■ رجیستر - مثل AX

■ حالت Immediate address - مثل 12H

■ آدرس دهی حافظه:

■ Direct addressing - مثل [3965]

■ Register indirect - مثل [BX]

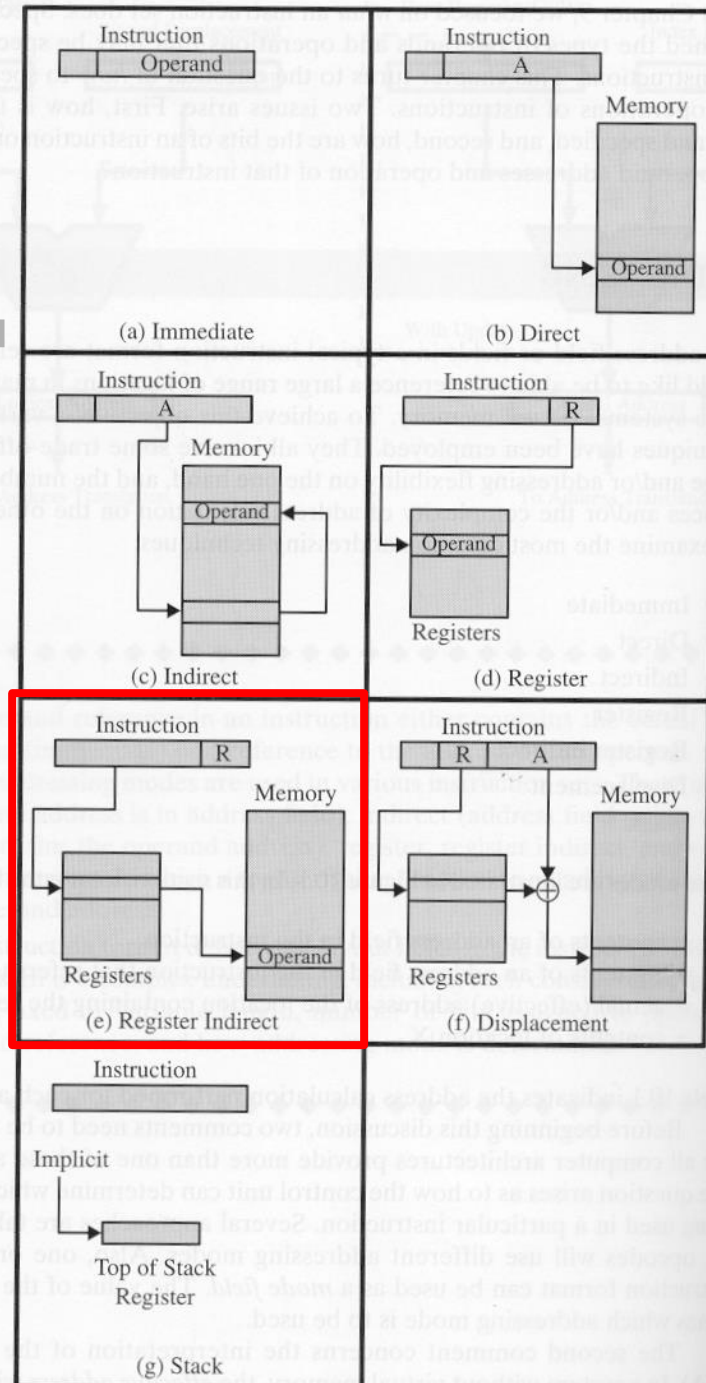
■ Based relative addressing

مثل: [BX+6], [BP]-10

■ Indexed relative addressing

مثل: [SI+5], [DI]-8

■ Based indexed addressing



آدرس دهی غیر مستقیم توسط رجیستر (Register indirect addressing)

■ آدرس مکانی از حافظه که رجیستری که عملوند است، آن آدرس را در خود نگه میدارد.

■ رجیسترهایی که برای این منظور استفاده میشوند: SI و DI و BX.

■ مثال:

```
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
```

حالت های آدرس دهی

انواع عملوندها در یک دستورالعمل:

■ رجیستر - مثل AX

■ حالت Immediate address - مثل 12H

■ آدرس دهی حافظه:

■ Direct addressing - مثل [3965]

■ Register indirect - مثل [BX]

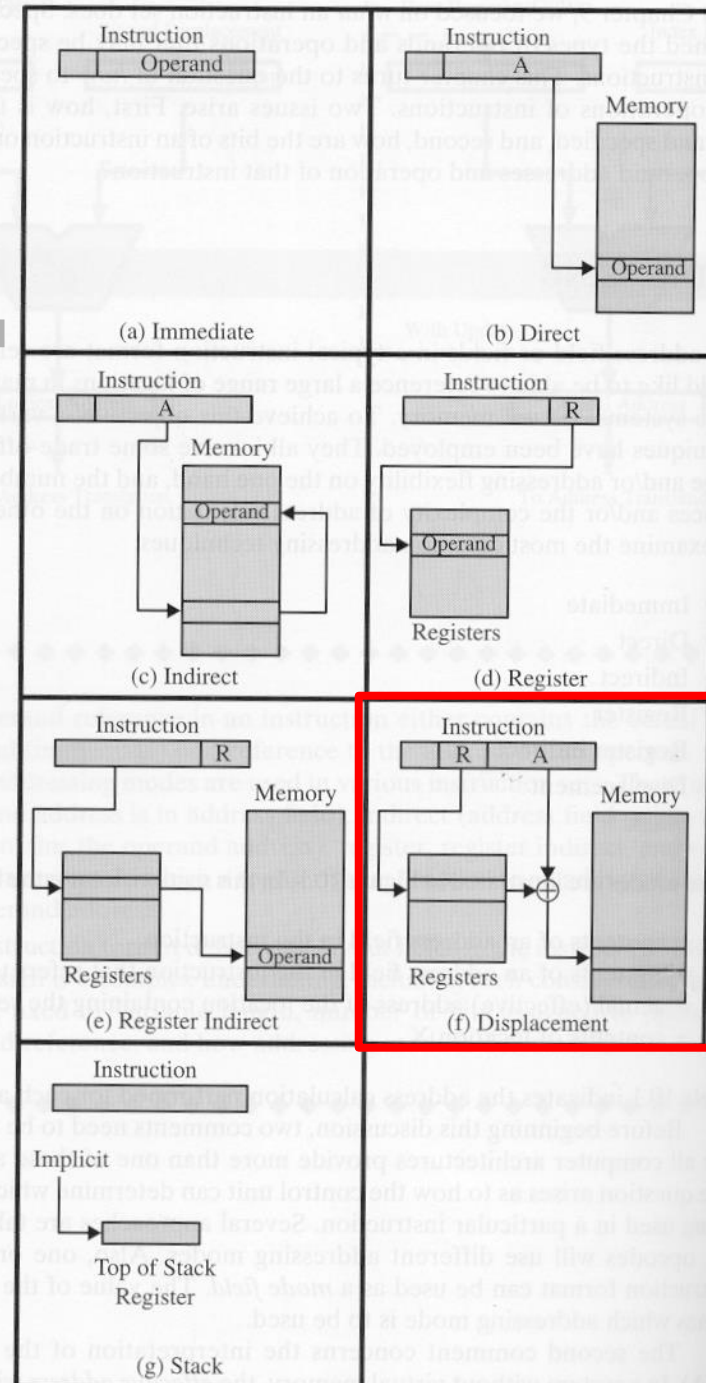
■ Based relative addressing

مثل: [BX+6], [BP]-10

■ Indexed relative addressing

مثل: [SI+5], [DI]-8

■ Based indexed addressing



آدرس دهی نسبی با رجیستر پایه (Based relative addressing)

- در این حالت، رجیستر پایه BX و BP و همچنین یک مقدار جابه جایی (displacement) استفاده میشوند تا آدرس مؤثر محاسبه شود.

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

- MOV CX,[BX]+10
10[BX]
[BX+10]

حالت های آدرس دهی

انواع عملوندها در یک دستورالعمل:

■ رجیستر - مثل AX

■ حالت Immediate address - مثل 12H

■ آدرس دهی حافظه:

■ Direct addressing - مثل [3965]

■ Register indirect - مثل [BX]

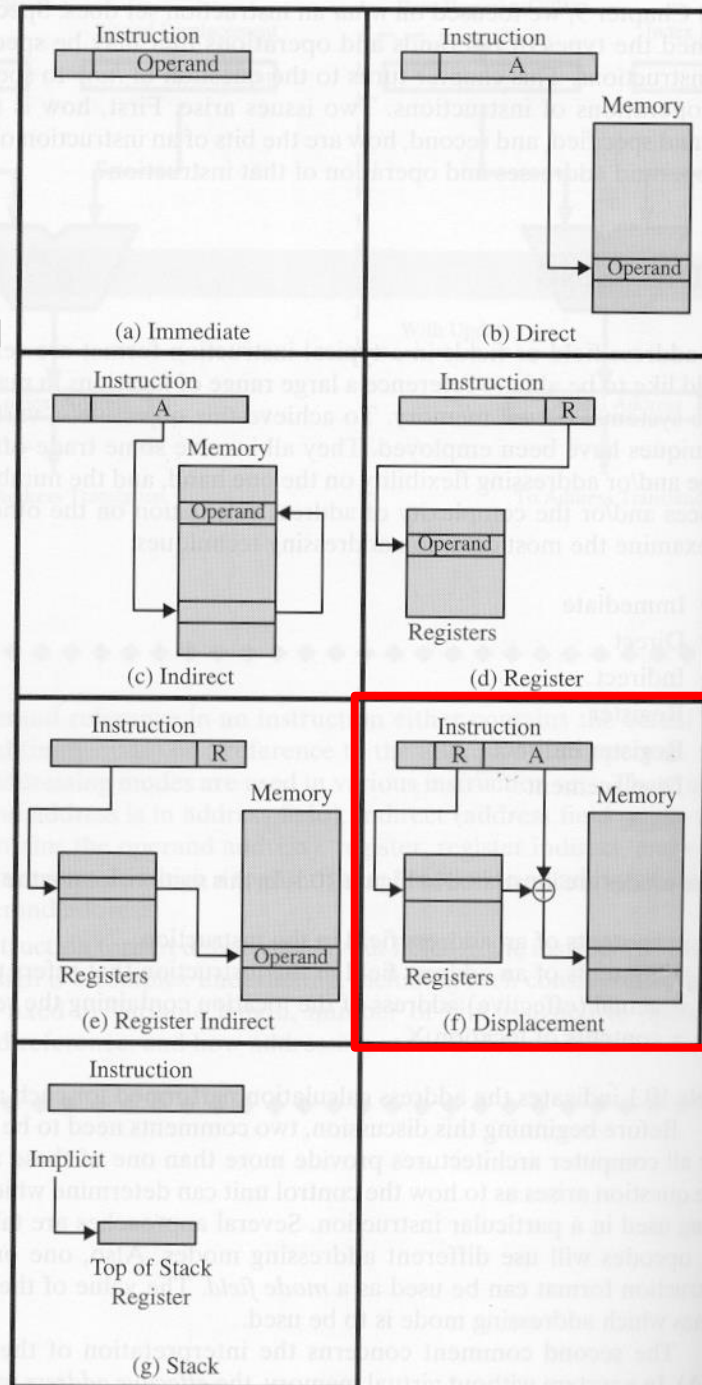
■ Based relative addressing

مثل: [BX+6], [BP]-10

■ Indexed relative addressing

مثل: [SI+5], [DI]-8

■ Based indexed addressing



آدرس دهی نسبی با رجیستر ایندکس (Indexed relative addressing)

■ در این حالت مشابه حالت قبلی کار میکند، به جز اینکه رجیسترهای SI و DI آدرس **offset** را نگه میدارند.

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

■ `MOV CX,[BX]+10`
`10[BX]`
`[BX+10]`

حالت های آدرس دهی

انواع عملوندها در یک دستورالعمل:

■ رجیستر - مثل AX

■ حالت Immediate address - مثل 12H

■ آدرس دهی حافظه:

■ Direct addressing - مثل [3965]

■ Register indirect - مثل [BX]

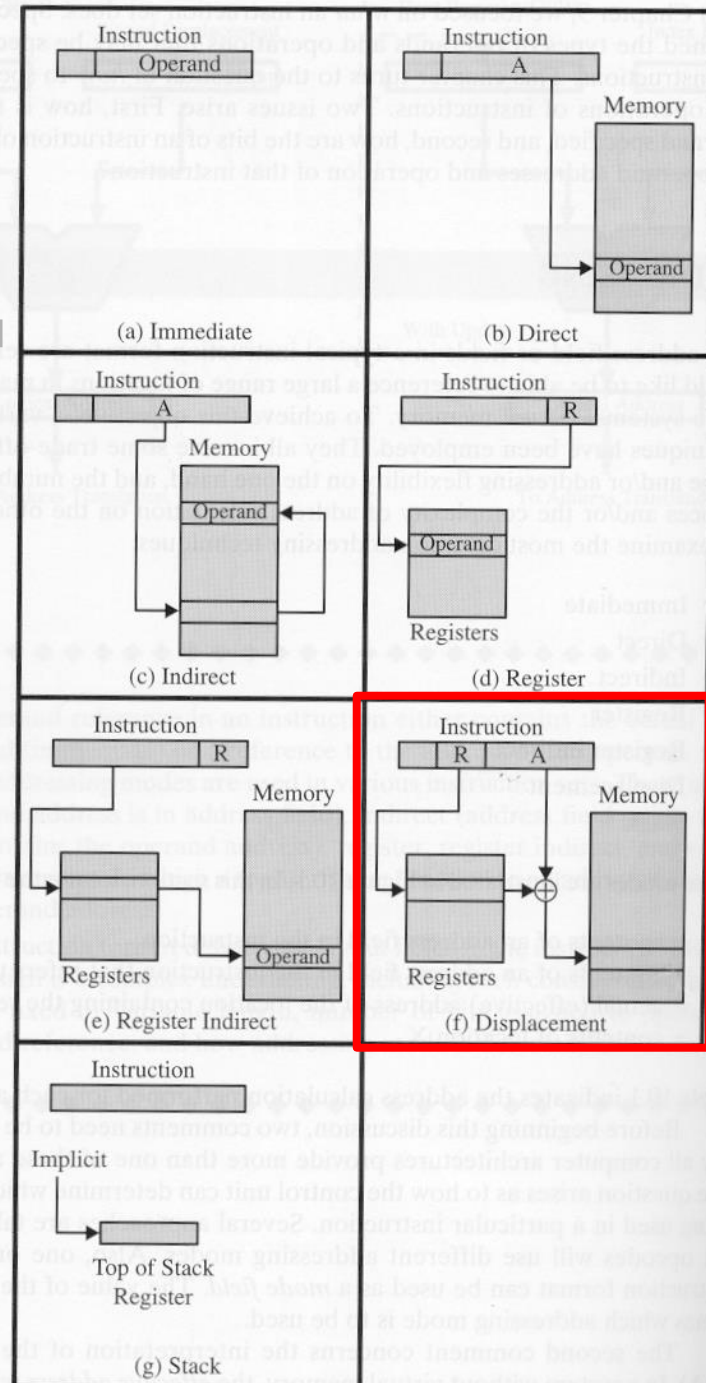
■ Based relative addressing

مثل: [BX+6], [BP]-10

■ Indexed relative addressing

مثل: [SI+5], [DI]-8

■ Based indexed addressing



آدرس دهی نسبی با ثبات پایه و ایندکس (Based Indexed addressing)

- ترکیبی از حالت آدرس دهی نسبی با رجیستر پایه و رجیستر ایندکس است.
- مثال:

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]

or

MOV AH,[SI+BP+29] ;the register order does not matter.

لغو کردن حالت رجیسترهای Offset پیشفرض (Overriding default offset register)

■ رجیسترهای Offset پیشفرض برای سگمنتهای مختلف بصورت زیر است:

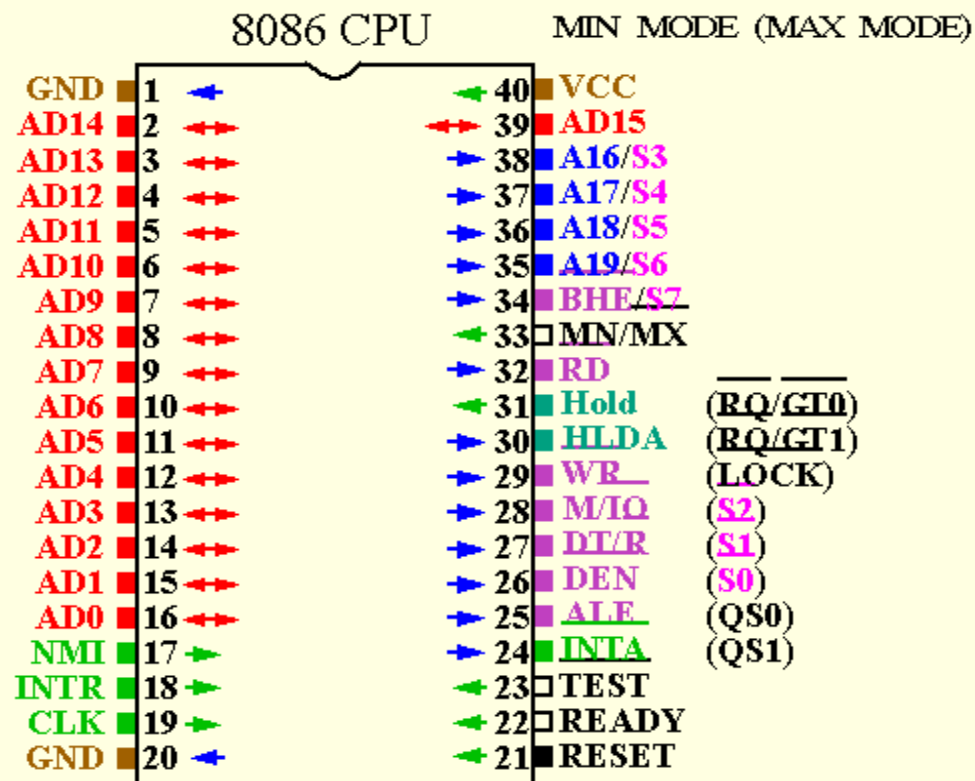
رجیسترهای سگمنت	CS	DS	ES	SS
رجیسترهای Offset	IP	SI , DI , BX	SI , DI , BX	SP , BP

■ پردازنده های 80x86 اجازه میدهند به برنامه نویس تا بتواند این پیشفرضها را لغو کرده و خودش رجیسترهای Offset دیگری را تعیین کند.

■ برای مثال:

دستورالعمل	سگمنت استفاده شده	سگمنت پیشفرض
MOV AX,CS:[BP]	CS:BP	SS:BP
MOV DX,SS:[SI]	SS:SI	DS:SI
MOV AX,DS:[BP]	DS:BP	SS:BP
MOV CX,ES:[BX]+12	ES:BX+12	DS:BX+12
MOV SS:[BX][DI]+32,AX	SS:BX+DI+32	DS:BX+DI+32

طرح پایه های پردازنده ۸۰۸۶ اینتل





GOOD LUCK