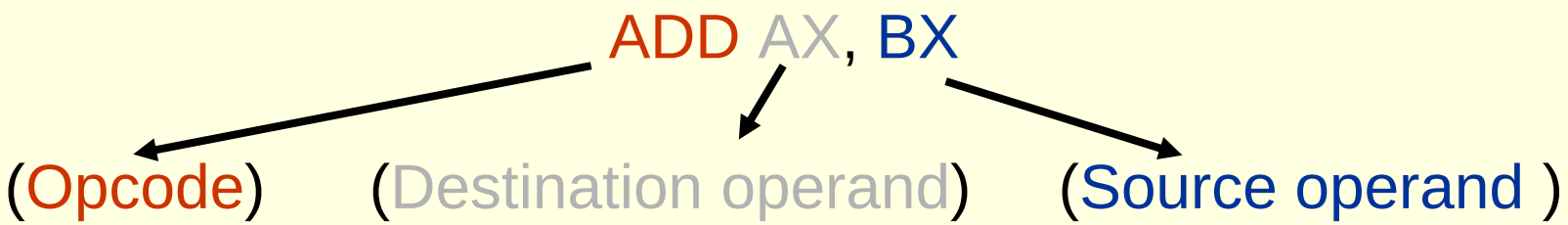
8086/8088 Instruction Set

Mohsen Nickray

Software

- The sequence of commands used to tell a microcomputer what to do is called a *program*,
- Each command in a program is called an *instruction*
- 8088 understands and performs operations for 117 basic instructions
- The native language of the IBM PC is the *machine language* of the 8088
- A program written in machine language is referred to as machine code
- In 8088 *assembly language*, each of the operations is described by alphanumeric symbols instead of 0-1s.



Instructions

LABEL: INSTRUCTION ; COMMENT

Address identifier code

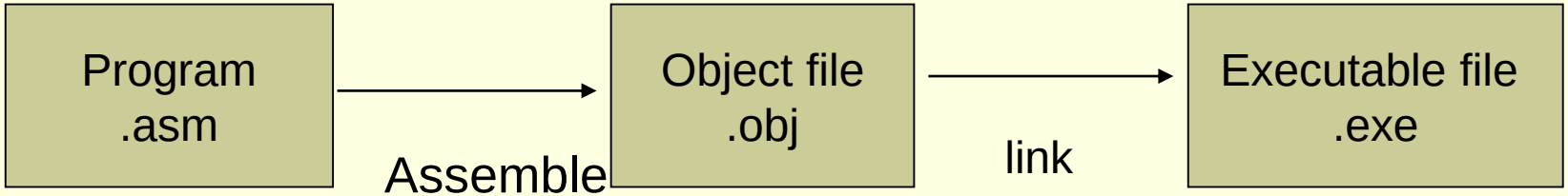
Does not generate any machine

- Ex. START: MOV AX, BX ; copy BX into AX
- There is a one-to-one relationship between assembly and machine language instructions
- A compiled machine code implementation of a program written in a high-level language results in inefficient code
  - More machine language instructions than an assembled version of an equivalent handwritten assembly language program

- Two key benefits of assembly language programming
  - It takes up less memory
  - It executes much faster

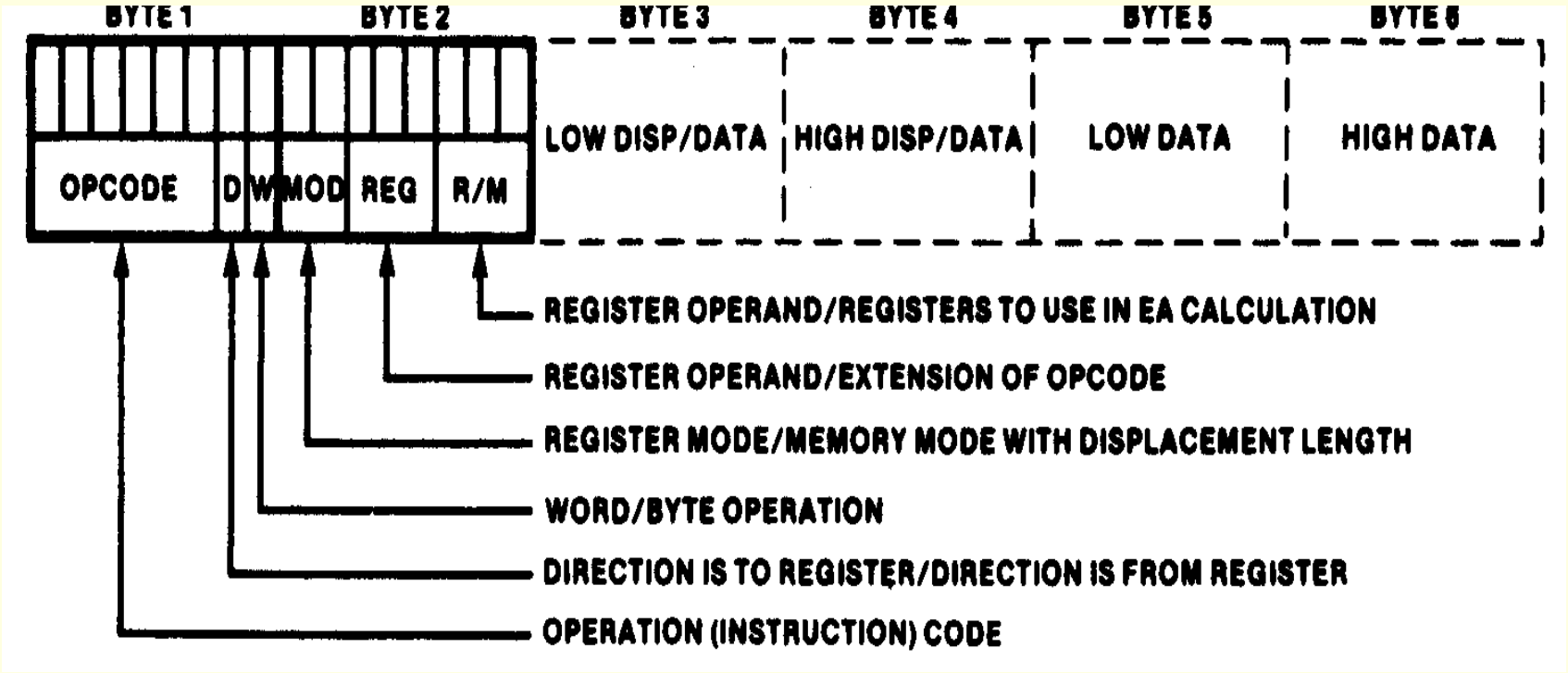
Flow of program development

The executable program could be .com, .exe, or .bin files



قالب دستورالعمل

- قالب عمومی دستور ها برای کد ماشین

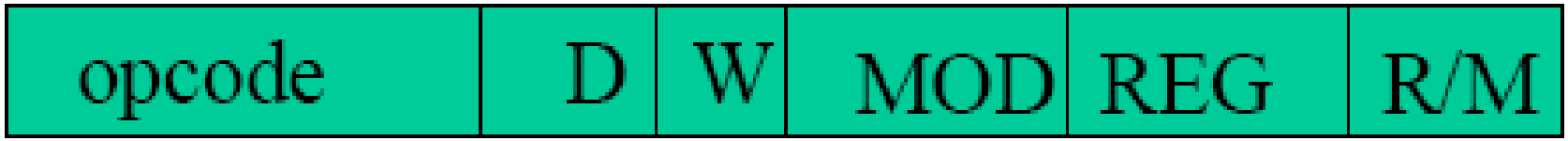


Converting Assembly Language Instructions to Machine Code

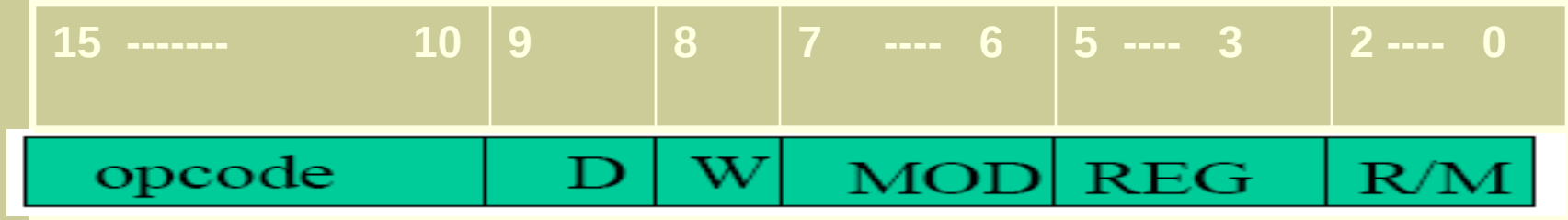
- An instruction can be coded with 1 to 6 bytes
- Byte 1 contains three kinds of information
  - – Opcode field (6 bits) specifies the operation (add, subtract, move)
  - – Register Direction Bit (D bit) Tells the register operand in REG field in byte 2 is source or destination operand

    1: destination 0: source
  - -Data Size Bit (W bit) Specifies whether the operation will be performed on 8-bit or 16-bit data

    0: 8 bits 1: 16 bits



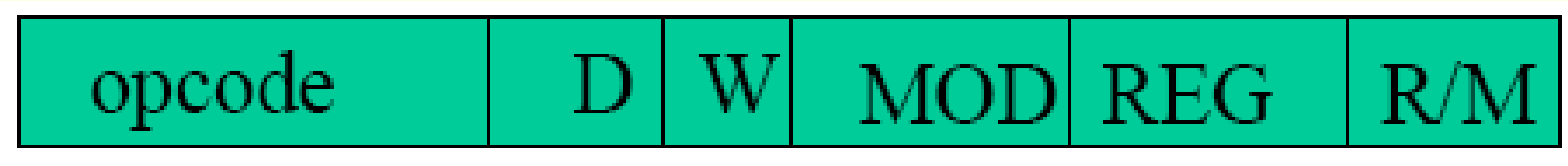
Field size



Byte 2 has three fields

- Mode field (MOD)
- Register field (REG) used to identify the register for the first operand
- Register/memory field (R/M field)

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



2-bit MOD field and 3-bit

Mode Field encoding

CODE	EXPLANATION
00	Memory Mode, no displacement follows*
01	Memory Mode, 8-bit displacement follows
10	Memory Mode, 16-bit displacement follows
11	Register Mode (no displacement)

*Except when R/M = 110, then 16-bit displacement follows

(a)

R/M field together specify the second operand

MOD = 11			EFFECTIVE ADDRESS CALCULATION			
R/M	W = 0	W = 1	R/M	MOD = 00	MOD = 01	MOD = 10
000	AL	AX	000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16
001	CL	CX	001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) + D16
010	DL	DX	010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16
011	BL	BX	011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16
100	AH	SP	100	(SI)	(SI) + D8	(SI) + D16
101	CH	BP	101	(DI)	(DI) + D8	(DI) + D16
110	DH	SI	110	DIRECT ADDRESS	(BP) + D8	(BP) + D16
111	BH	DI	111	(BX)	(BX) + D8	(BX) + D16

(b)

Register/memory (R/M) Field Encoding

قالب دستور العمل

- *مثال:*
  - MOV BL, AL
    - دستور بالا را در ماشین کد به رمز در آورید
- *حل مساله:*
  - OPCODE = 100010 (for MOV), D = 0 (source), W = 0 (8-bit)
  - در نتیجه BYTE 1 = $10001000_2 = 88_{16}$
- عملوند دوم در فیلد reg نشان داده شده است.
  - REG = 000, MOD = 11, R/M = 011
  - بنابراین , BYTE 2 = $11000011_2 = C3_{16}$
  - MOV BL, AL = $88C3_{16}$

Examples

MOV BL,AL ($88C3_{16}$)

Opcode for MOV = 100010

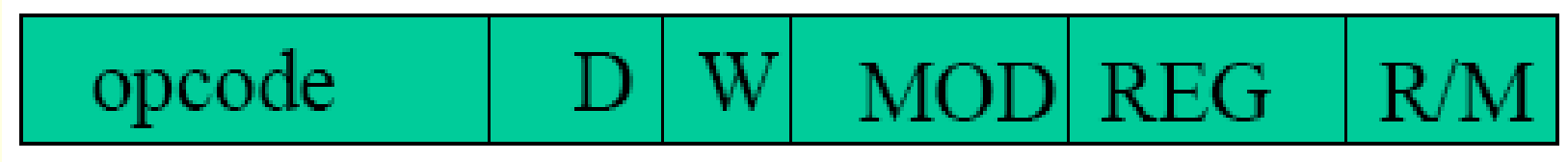
D = 0 (AL source operand)

W bit = 0 (8-bits)

Therefore byte 1 is $10001000_2=88_{16}$

- MOD = 11 (register mode)
- REG = 000 (code for AL)
- R/M = 011 (destination is BL)

Therefore Byte 2 is $11000011_2=C3_{16}$



قالب دستورالعمل

- *مثال*
  - ADD AX, [SI]
    - دستور بالا را در ماشین کد به رمز در آورید
- *حل مساله:*
  - OPCODE = 000000 (for ADD), D = 1 (dest.), W = 1 (16-bit)
  - در نتیجه $\text{BYTE 1} = 00000011_2 = 03_{16}$
- در بایت دوم عملوند مقصد که به وسیله REG شناسایی میشود در AX است
  - REG = 000, MOD = 00, R/M = 100
  - در نتیجه $\text{BYTE 2} = 00000100_2 = 04_{16}$
  - $\text{ADD AX, [SI]} = 0304_{16}$

قالب دستورالعمل

- *مثال*
  - XOR CL, [1234H]
    - دستور بالا را در ماشین کد به رمز در آورید
- *حل مساله:*
  - OPCODE = 001100 (for XOR), D = 1 (dest.), W = 0 (8-bit)
  - در نتیجه $BYTE\ 1 = 00110010_2 = 32_{16}$
- در بایت دوم عملوند مقصد که به وسیله REG شناسایی میشود در CL است
  - REG = 001, MOD = 00, R/M = 110
  - بنابراین $BYTE\ 2 = 00001110_2 = 0E_{16}$
  - $BYTE\ 3 = 34_{16}$ $BYTE\ 4 = 12_{16}$
  - $XOR\ CL, [1234H] = 320E3412_{16}$

قالب دستور العمل

- *مثال*
  - ADD [BX][DI]+1234H, AX
    - دستور بالا را در ماشین کد به رمز در آورید
- *حل مساله:*
  - OPCODE = 000000 (for ADD), D = 0 (source), W = 1 (16-bit)
  - در نتیجه BYTE 1 = $00000001_2 = 01_{16}$
- در بایت دوم عملوند مقصد که به وسیله REG شناسایی میشود در AX است
  - REG = 000, MOD = 10, R/M = 001
  - بنابراین BYTE 2 = $10000001_2 = 81_{16}$
    BYTE 3 = 34_{16} BYTE 4 = 12_{16}
  - ADD [BX][DI]+1234H, AX = 01813412_{16}

قالب دستورالعمل

- *مثال*
  - MOV WORD PTR [BP][DI]+1234H, 0ABCDH
    - دستور بالا را در ماشین کد به رمز در آورید
  - *حل مساله:*
    - این مثال قالب عمومی را دنبال نمیکند.
  - MOV -> 1100011W, and W = 1 for word-size data
  - BYTE 1 = 11000111_2 = $C7_{16}$
  - BYTE 2 = (MOD)000(R/M) = 10000011_2 = 83_{16}
  - BYTE 3 = 34_{16} BYTE 4 = 12_{16}
  - BYTE 5 = CD_{16} BYTE 6 = AB_{16}
  - MOV WORD PTR [BP][DI]+1234H, 0ABCDH = $C7833412CDAB_{16}$

8086 Instructions

AAA	CMPSB	IRET	JNAE	JP	LOOPZ	PUSHF	SBB
AAD	CMPSW	JA	JNB	JPE	MOV	RCL	SCASB
AAM	CWD	JAE	JNBE	JPO	MOVSB	RCR	SCASW
AAS	DAA	JB	JNC	JS	MOVSW	REP	SHL
ADC	DAS	JBE	JNE	JZ	MUL	REPE	SHR
ADD	DEC	JC	JNG	LAHF	NEG	REPNE	STC
AND	DIV	JCXZ	JNGE	LDS	NOP	REPNZ	STD
CALL	HLT	JE	JNL	LEA	NOT	REPZ	STI
CBW	IDIV	JG	JNLE	LES	OR	RET	STOSB
CLC	IMUL	JGE	JNO	LODSB	OUT	RETF	STOSW
CLD	IN	JL	JNP	LODSW	POP	ROL	SUB
CLI	INC	JLE	JNS	LOOP	POPA	ROR	TEST
CMC	INT	JMP	JNZ	LOOPE	POPF	SAHF	XCHG
CMP	INTO	JNA	JO	LOOPNE	PUSH	SAL	XLATB
				LOOPNZ	PUSHA	SAR	XOR

دستورالعمل های 8086

- به 7 طبقه تقسیم می شود:
  - انتقال داده
  - محاسبات
  - منطقی
  - کنترلی
  - دستورالعمل های پردازنده کنترلی
  - کارهای حلقه
  - کنترل وقفه (interrupt)

انتقال داده

- عملگرهای انتقال داده شامل دستورهای زیر است:
  - MOV (انتقال بایت یا ورد)
  - XCHG (جابه جایی بایت یا ورد)
  - XLAT (ترجمه بایت)
  - LEA (بارگذاری آدرس موثر)
  - LDS (بارگذاری سگمنت داده)
  - LES (بارگذاری سگمنت اضافی)
  - PUSH Src: ورد را در پشته کپی میکند
  - POP dest: ورد را از پشته در رجیستر مقصد کپی میکند.
  - IN acc, port :

    8 یا 16 بیت داده را از درگاه در انباشته گر (accumulator) کپی میکند
  - OUT port, acc

دستورهای انتقال داده

- دستورالعمل MOV
- دستورالعمل MOV برای انتقال یک بایت یا ورد داده از عملوند مبدا به عملوند مقصد استفاده میشود.

کد	معنی	قالب	عملکرد	تاثیر بر FLAG
MOV	انتقال دادن	MOV D,S	(S) →(D)	هیچ

- **Examples:**

MOV CX, 037AH ;

037AH را به CX انتقال بده

MOV AX, BX ;

محتوای رجیستر BX را به AX کپی کن

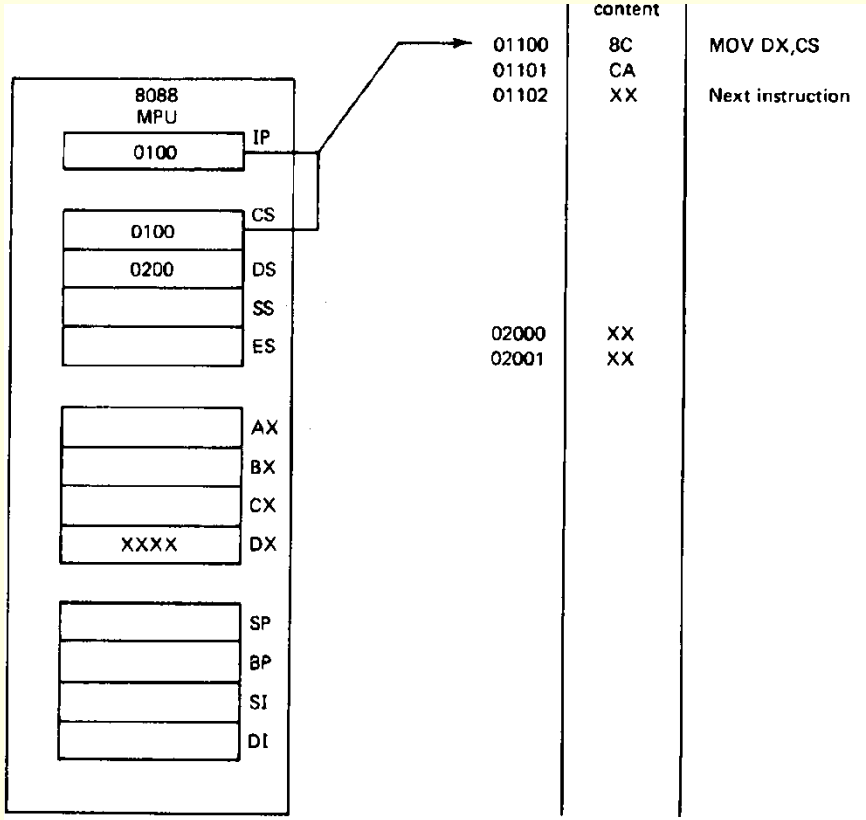
MOV DL,[BX] ;

یک بایت از حافظه ر که BX محتوی آدرس OFFSET آن است را در DL کپی کن

ساختار انتقال داده

- دستورالعمل MOV

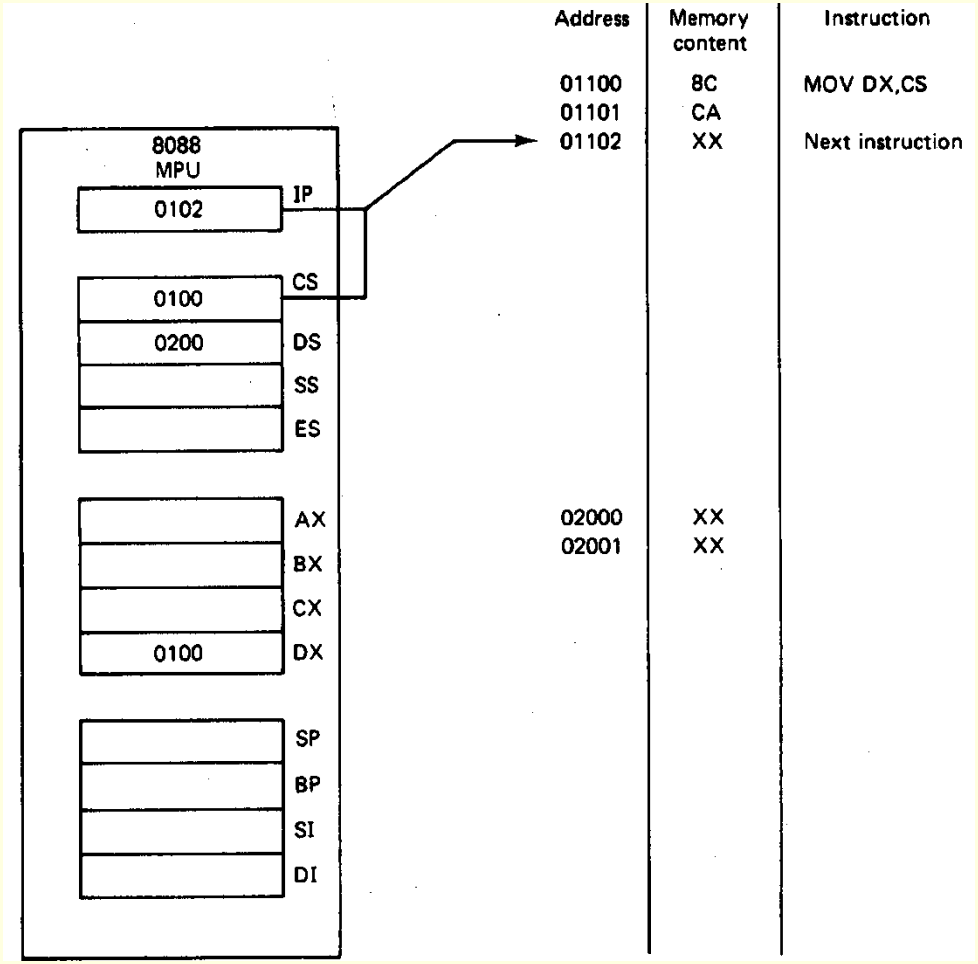
- MOV DX, CS



ساختار انتقال داده

- دستورالعمل MOV

- MOV DX, CS



ساختار انتقال داده

- *مثال*

انجام دستورالعمل زیر چه تاثیری دارد ؟

MOV CX, [SOURCE_MEM]

در حالیکه SOURCE-MEM برابر با 20_{16} است و محتویات DS برابر $1A00_{16}$

- *راه حل :*

$((DS)0+20_{16})$ $\longrightarrow$ (CL)

$((DS)0+20_{16}+1_{16})$ $\longrightarrow$ (CH)

بنابراین CL با محتوای حافظه به آدرس زیر بارگذاری میشود

$1A000_{16} + 20_{16} = 1A020_{16}$

CX با محتوای حافظه به آدرس زیر بارگذاری میشود

$1A000_{16} + 20_{16} + 1_{16} = 1A021_{16}$

دستورالعمل های MOVSW/MOVSB

- بایت/ورد واقع در در DS:[SI] را در ES:[DI] کپی کن و مقدار SI و DI را جدید کن.

```
• ES:[DI] = DS:[SI]
• if DF = 0 then
    o SI = SI + 1
    o DI = DI + 1
  else
    o SI = SI - 1
    o DI = DI - 1
```

- مثال

```
ORG 100h

CLD
LEA SI, a1
LEA DI, a2
MOV CX, 5
REP MOVSB

RET

a1 DB 1,2,3,4,5
a2 DB 5 DUP(0)
```

ساختار انتقال داده

- دستورالعمل XCHG
- دستورالعمل عوض کردن XCHG برای عوض کردن داده بین دو رجیستر یا بین یک رجیستر و مجتوای خانه ای از حافطه مورد استفاده قرار میگیرد.

e.g. XCHG AX, DX

کد	معنی	قالب	عملکرد	تاثیر بر FLAG
XCHG	جابه جایی	XCHG D,S	(D) ←→ (S)	هیچ

مبدا	مقصد
رجیستر	حافظه
رجیستر	رجیستر
حافظه	رجیستر

ساختار انتقال داده

- *مثال:*

نتیجه اجرای دستورالعمل زیر چیست ؟

XCHG [SUM], BX

در حالیکه $SUM = 1234_{16}$, $(DS)=1200_{16}$

- *راه حل*

$((DS)0+SUM) \longleftrightarrow (BX)$

$PA = 12000_{16} + 1234_{16}=13234_{16}$

- اجرای دستورالعمل موجب جا به جایی 16 بیتی زیر میشود

$(13234_{16}) \longleftrightarrow (BL)$

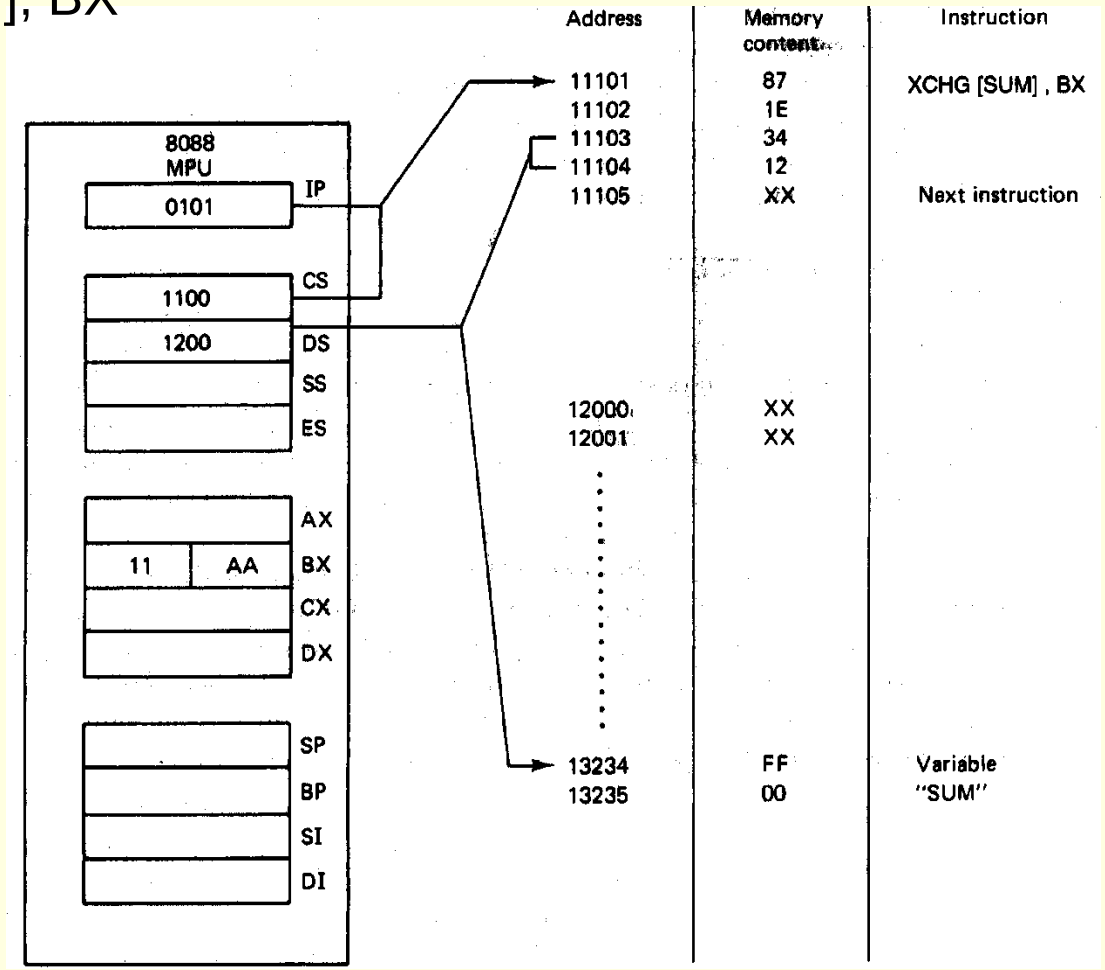
$(13235_{16}) \longleftrightarrow (BH)$

در نتیجه $(BX) = 00FF_{16}$, $(SUM) = 11AA_{16}$

ساختار انتقال داده

- دستورالعمل XCHG

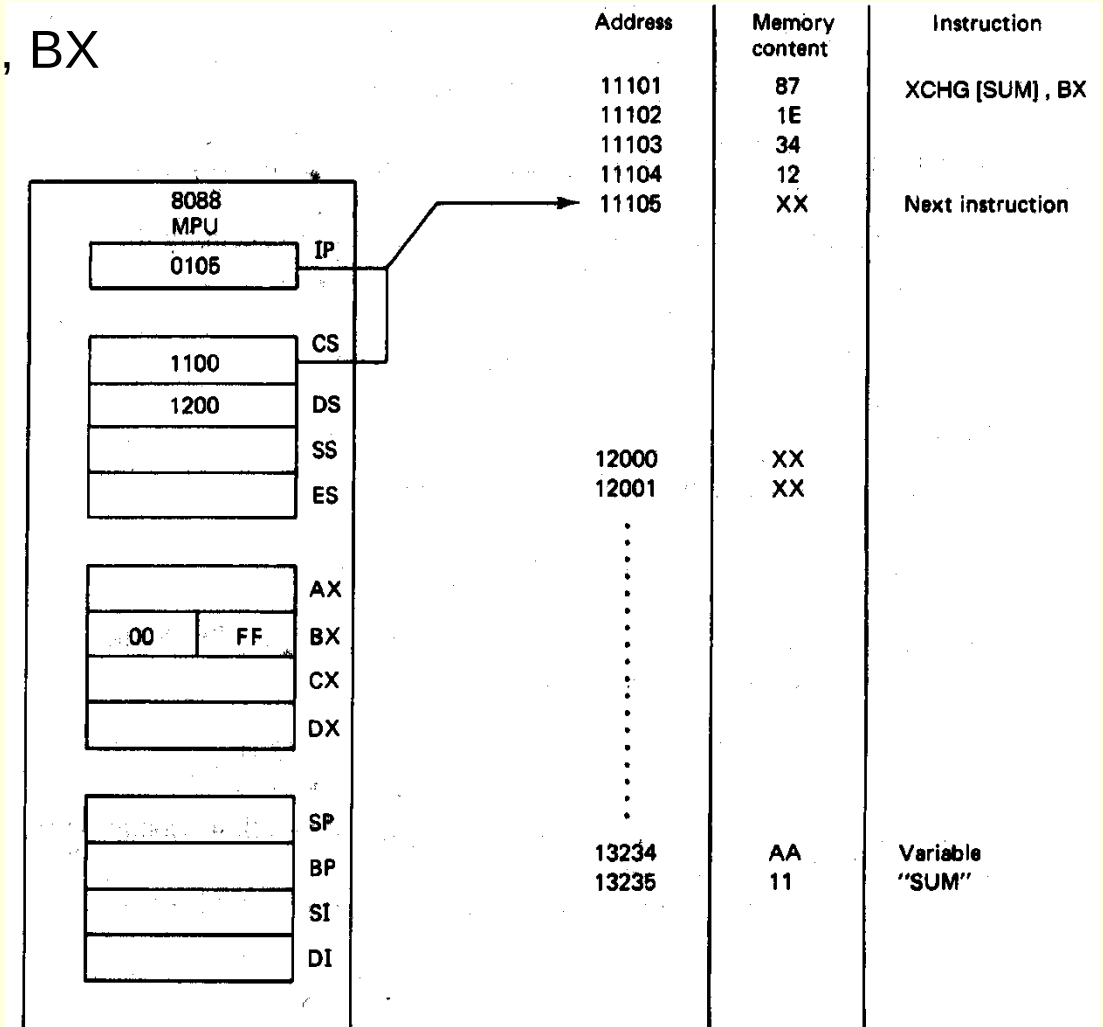
- XCHG [SUM], BX



ساختار انتقال داده

- دستورالعمل XCHG

- XCHG [SUM], BX



دستور العمل IN

- **IN** AL , im.byte
  AX , im.byte
  AL , DX
  AX , DX

- Input from port into **AL** or **AX**.
  - Second operand is a port number
  - If required to access port number over 255 - **DX** register should be used.

IN - مثال

- **IN AL, 0C8H**
- **IN AX, 34H**
- **MOV DX, 0FF78H**
- **IN AL, DX**
- **IN AX, DX**

دستور العمل OUT

- **OUT** im.byte,AL
  im.byte, AX
  DX, AL
  DX, AX

- Output from **AL** or **AX** to port.
  - First operand is a port number
  - If required to access port number over 255 - **DX** register should be used.

OUT - مثال

- **OUT 3BH, AL**
- **OUT 2CH,AX**

- **MOV DX, 0FFF8H**
- **OUT DX, AL**
- **OUT DX, AX**

دستور العمل PUSH

- **PUSH** REG | SREG | memory | immediate

  Store 16 bit value in the stack.

- Algorithm:

  SP = SP - 2

  SS:[SP] (top of the stack) = operand

- مثال

PUSH BX دو تا از SP کم میکند و BX را در پشته کپی میکند

PUSH DS دو تا از SP کم میکند و DS را در پشته کپی میکند

PUSH TABLE[BX]

دو تا از SP کم میکند و یک ورد از حافظه واقع در EA= TABLE + [BX] را در پشته کپی میکند

PUSHA دستورالعمل

- PUSHA (No operands)

  Push all general purpose registers AX, CX, DX, BX, SP, BP, SI, DI in the stack.

- Algorithm:

```
PUSH AX
PUSH CX
PUSH DX
PUSH BX
PUSH SP
PUSH BP
PUSH SI
PUSH DI
```

PUSHF دستور العمل

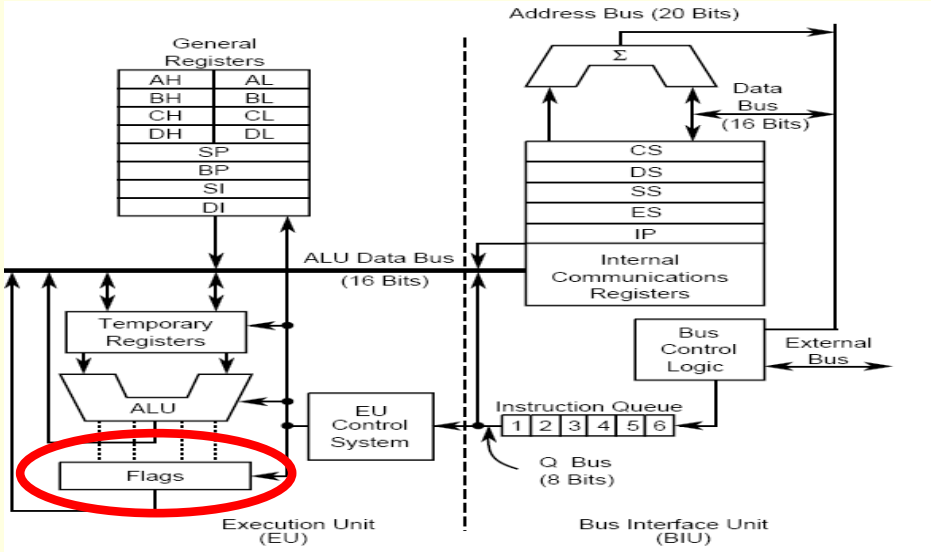
- PUSHF (No operands)

  Store flags register in the stack.

- Algorithm:

  SP = SP - 2

  SS:[SP] (top of the stack) = flags



دستور العمل POP

- POP REG | SREG | memory

  Get 16 bit value from the stack.

- Algorithm:
  operand = SS:[SP] (top of the stack)

  SP = SP + 2

مثال

POP DX

POP DS

POP TABLE [BX]

یک ورد از بالای پشته را در حافظه EA= TABLE + [BX] کپی میکند

POPA دستورالعمل

- POPA (No operands)

  Pop all general purpose registers DI, SI, BP, SP, BX, DX, CX, AX from the stack.

- Algorithm:

```
POP DI
POP SI
POP BP
POP xx (SP value ignored)
POP BX
POP DX
POP CX
POP AX
```

POPF دستور العمل

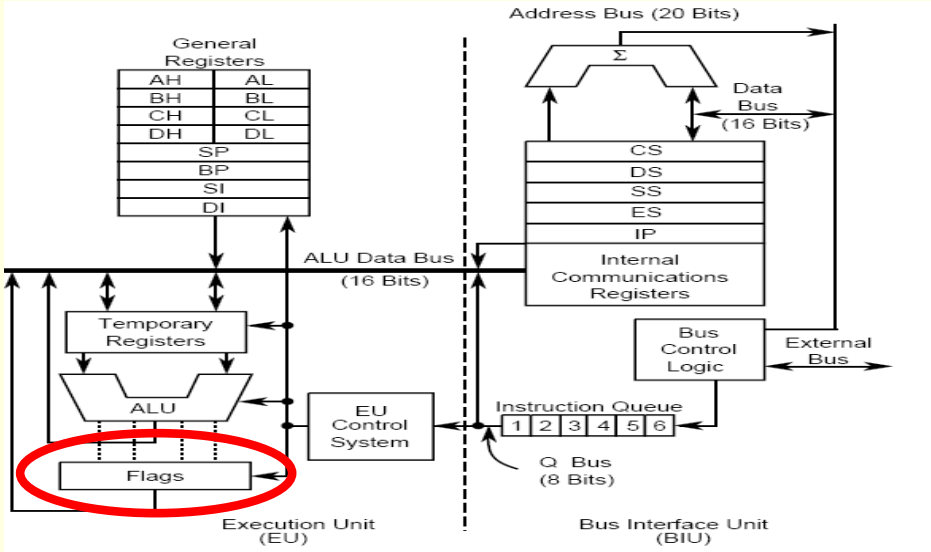
- POPF (No operands)

  Get flags register from the stack.

- Algorithm:

  flags = SS:[SP] (top of the stack)

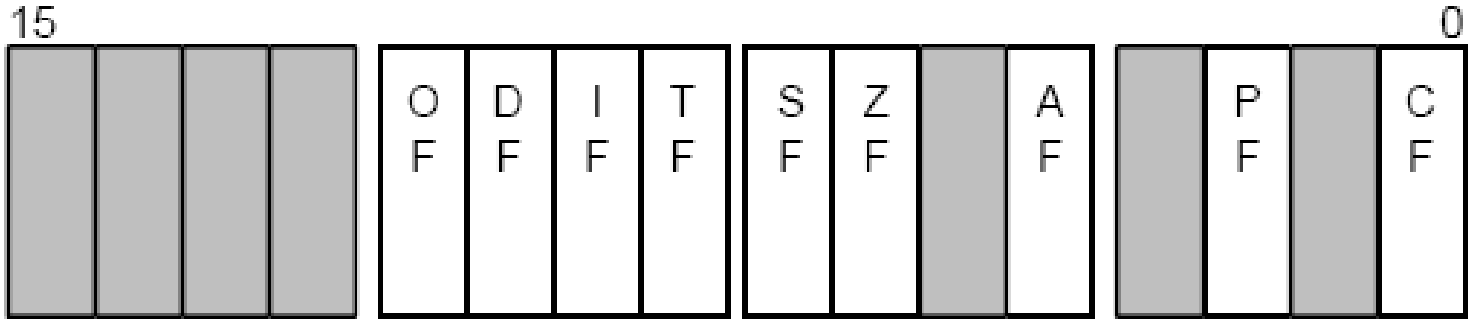
  SP = SP + 2



دستورالعمل LAHF و SAHF

- دستورالعمل LAHF – بایت پایینی از رجیستر flag را در AH کپی میکند
- Load AH Flags (in low position)
- دستورالعمل SAHF - AH را در بایت پایینی از رجیستر FLAG کپی میکند
- Save AH Flags (from low position)
- مثال

- دستورالعمل LAHF مقدار SF, ZF, AF, PF, CF را که به ترتیب بیت های 7, 6, 4,2,0 هستند را در AH کپی میکند.



دستورالعمل های LEA, LDS , LES

- دستورالعمل های LEA, LDS , LES قابلیت دسترسی به آدرسهای حافظه را به وسیله بارگذاری آدرس OFFSET 16 بیتی در یک رجیستر و یا یک رجیستر محتوی آدرس بخشی از DS یا ES فراهم میکند.

mnemonic	مفهوم	format	operation	تاثیر بر فلگ ها
LEA	افکتیو آدرس را بارگذاری کن	LEA Reg16,EA	EA→(REG16)	هیچ
LDS	DSورجیستر را بارگذاری کن	LDS Reg16,EA	EA→(REG16) EA+2→(DS)	هیچ
LES	ES و رجیستر را بارگذاری کن	LES Reg16,EA	EA→(REG16) EA+2→(ES)	هیچ

دستورالعمل LEA (load effective adrs)

- این دستورالعمل آدرس OFFSET یک متغیر یا خانه ای از حافظه که به عنوان مبدا نام گذاری شده را تعیین میکند و آدرس OFFSET را در رجیستر 16 بیتی تعیین شده میگذارد.

- **LEA BX, PRICE**

LEA CX, [BX][DI]

LDS(load data segment) دستورالعمل

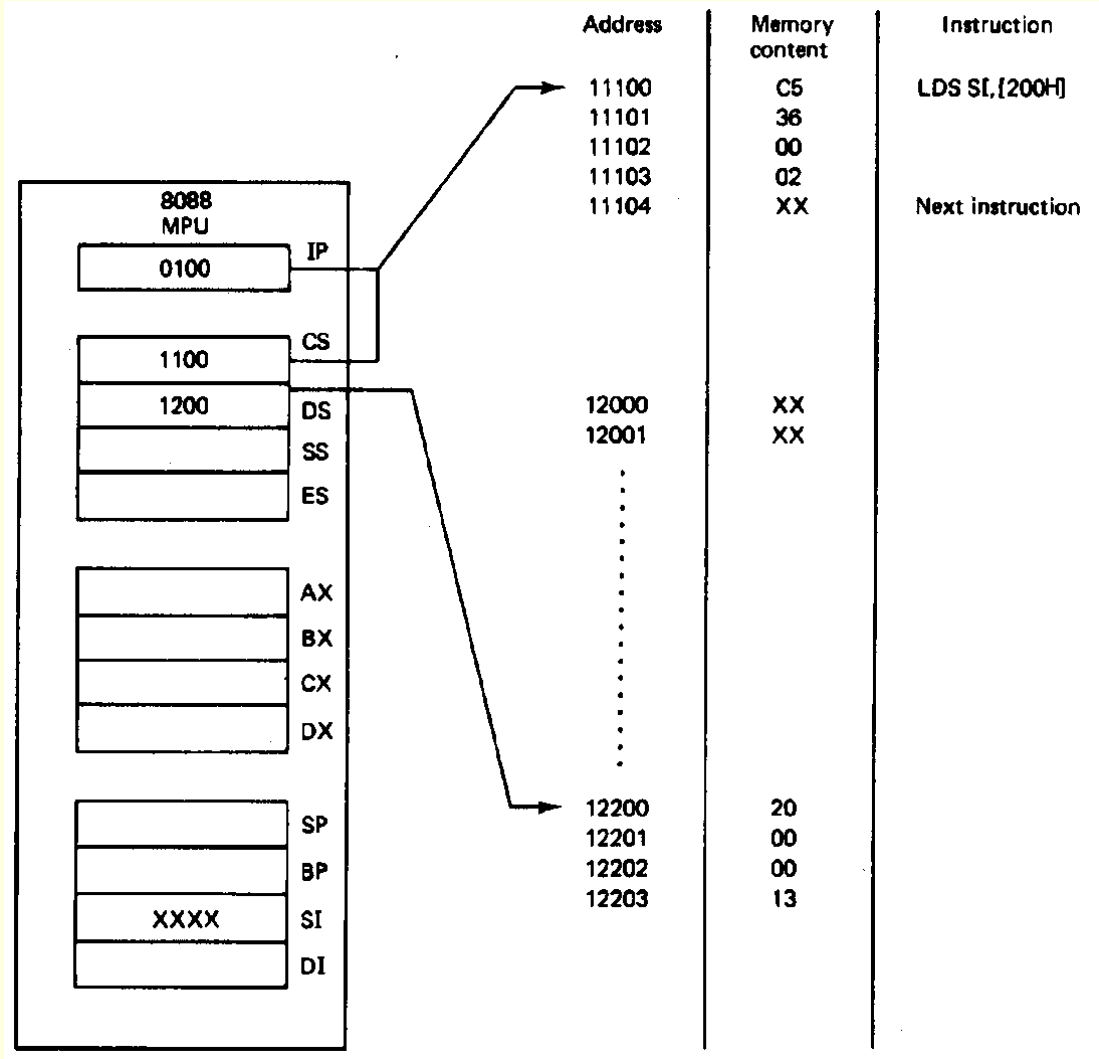
- **LDS BX, [4326]**

محتوای خانه حافظه واقع در 4326H در DS را در BL کپی میکند
و محتوای 4327H را در BH کپی میکند

محتوای 4328H و 4329H در DS را در DSR کپی میکند

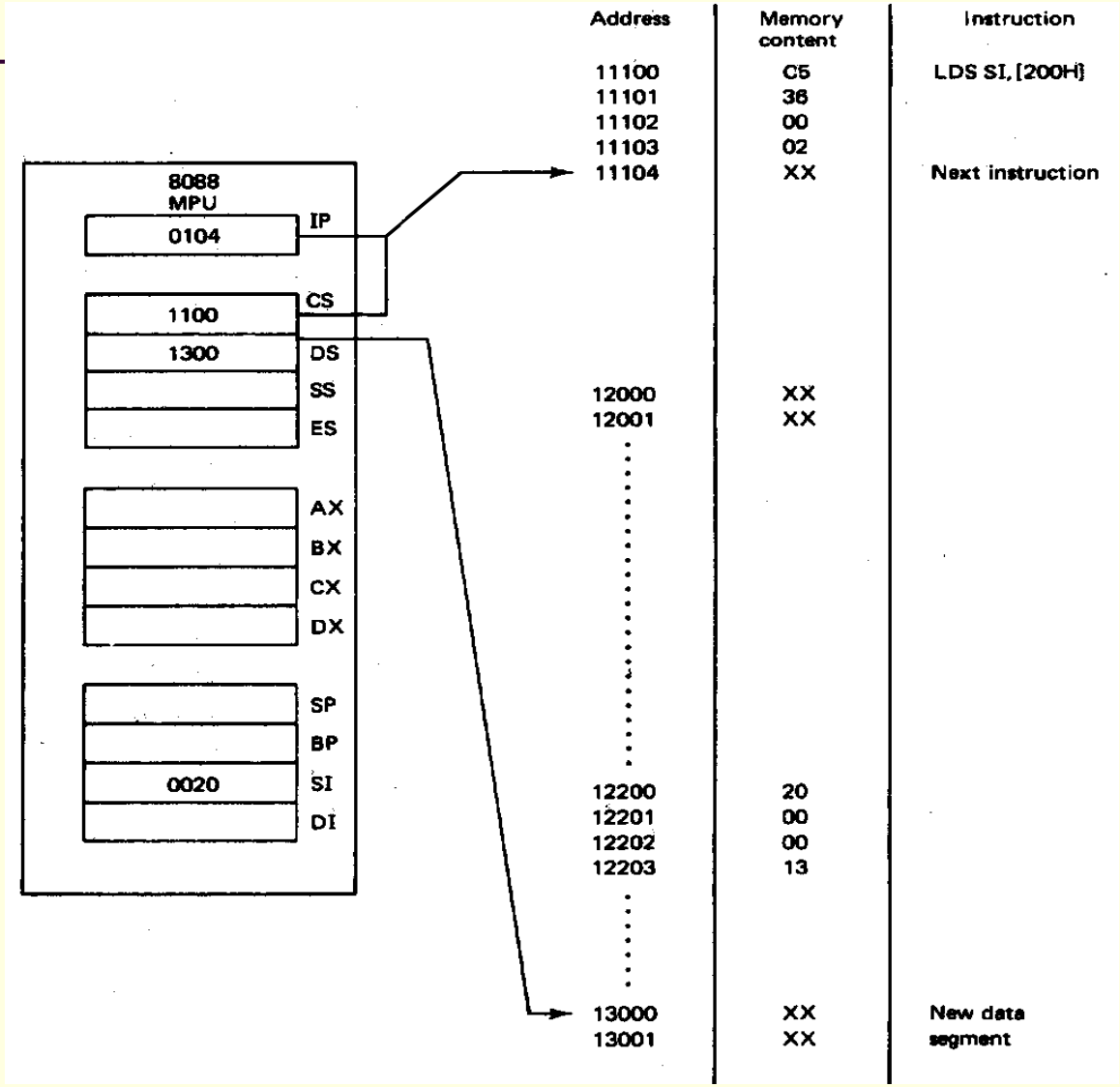
ساختار انتقال داده

- LDS SI, [200H]



ساختار انتقال داده

- LDS SI, [200H]



LES-load Extra Segment دستورالعمل

- **LES BX, [4326]**

محتوای خانه حافظه واقع در 4326H در ES را در BL کپی میکند
و محتوای 4327H را در BH کپی میکند

محتوای 4328H و 4329H در ES کپی میکند

دستورالعمل های محاسباتی

- دستورالعمل های محاسباتی شامل موارد زیر هستند:
  - جمع
  - تفریق
  - ضرب
  - تقسیم
- انواع داده ها:
  - بایت باینری بدون علامت
  - بایت باینری علامت دار
  - ورد باینری بدون علامت
  - ورد باینری علامت دار
  - بایت های دسیمال UNPACKED
  - بایت های دسیمال PACKED
  - عددهای ASCII

دستورالعمل های محاسباتی

جمع	
ADD	*بایت یا ورد را جمع کن*
ADC	*بایت یا ورد را با کری جمع کن*
INC	*بایت یا ورد را یکی اضافه کن*
AAA	*تنظیم ASCII* برای جمع
DAA	تنظیم ده تایی برای جمع
تفریق	
SUB	بایت یا ورد را تفریق کن
SBB	بایت یا ورد را با کری تفریق کن
DEC	بایت یا ورد را یکی کم کن
NEG	بایت یا ورد را منفی کن
AAS	تنظیم ASCII برای تفریق
DAS	تنظیم ده تایی برای تفریق
ضرب	
MUL	ضرب بایت یا ورد بدون علامت
IMUL	ضرب بایت یا ورد علامت دار
AAM	تنظیم ASCII برای ضرب
تقسیم	
DIV	تقسیم بایت یا ورد بدون علامت
IDIV	تقسیم بایت یا ورد علامت دار
AAD	تنظیم ASCII برای تقسیم
CBW	تبدیل بایت به ورد
CWD	تبدیل ورد به دابل ورد

دستورالعمل های محاسباتی

- دستورالعمل های جمع :
- ADD, ADC, INC, AAA, DAA

MNEMONIC	مفهوم	FORMAT	OPERATION	تاثیر بر فلگ ها
ADD	جمع کردن	ADD D,S	(S)+(D)→(D) CARRY→(CF)	OF,SF,ZF,AF,PF,CF
ADC	با کری جمع کردن	ADC D,S	(S)+(D)+(CF)→(D) CARRY→(CF)	OF,SF,ZF,AF,PF,CF
INC	یکی اضافه کن	INC D	(D)+1→(D)	OF,SF,ZF,AF,PF
AAA	تنظیم اسکی برای جمع	AAA		OF,SF,ZF,AF,PF,AF, CF, تعریف نشده
DAA	تنظیم ده تایی برای جمع	DAA		SF,ZF,AF,PF,CF,OF, تعریف نشده

دستورالعمل های محاسباتی

- دستورالعمل های جمع : ADD, ADC, INC, AAA, DAA

مبدا	مقصد
رجیستر	رجیستر
حافظه	رجیستر
رجیستر	حافظه
عدد ناگهانی	شمارشگر
عدد ناگهانی	رجیستر
عدد ناگهانی	جافظه

عملگرهای مجاز برای ADDوADC

مقصد
رجیستر 16 تایی
رجیستر 8 تایی
حافظه

عملگرهای مجاز برای INC

دستورالعمل های محاسباتی

- *مثال:*
- در نظر بگیرید که AX و BX دارای مقادیر به ترتیب 1100_{16} و $0ABC_{16}$ باشند نتیجه اجرای دستورالعمل ADD AX , BX چیست ؟
- *راه حل:*

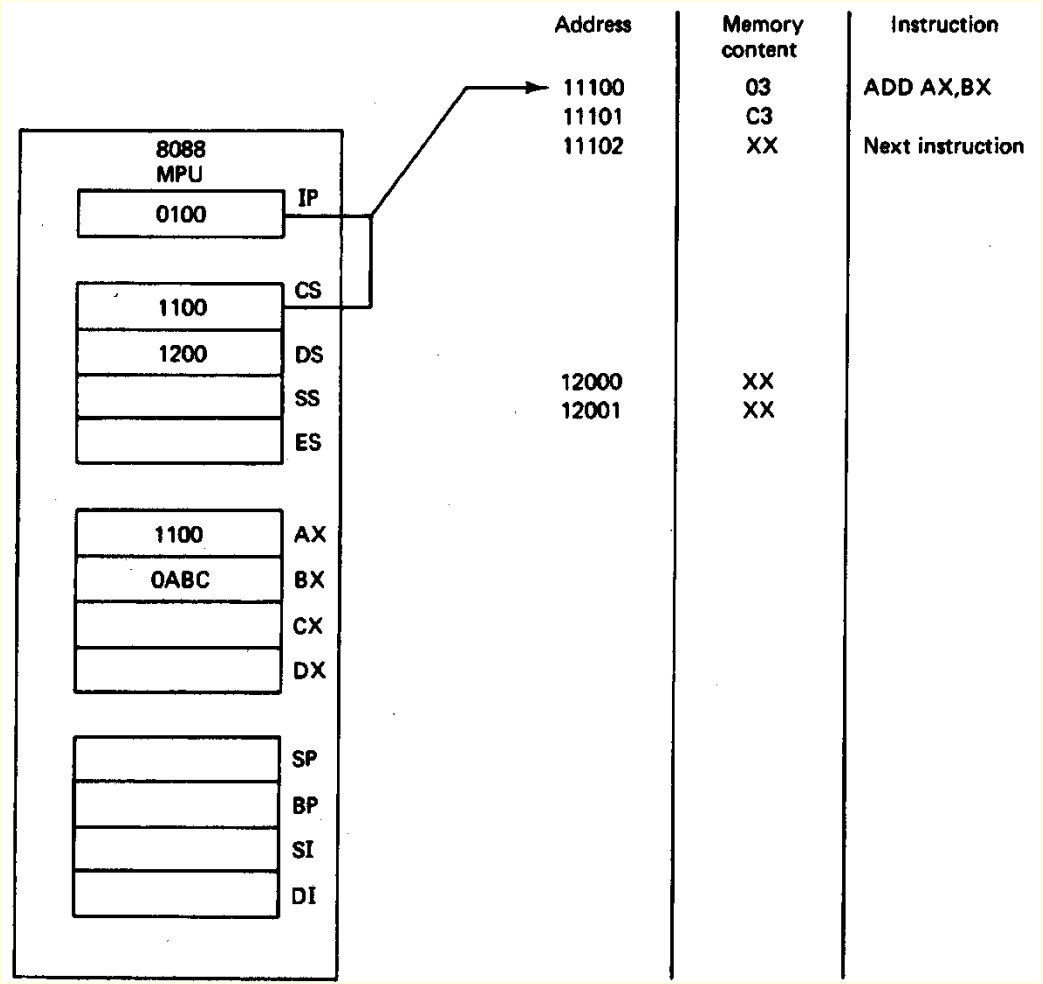
$$(BX)+(AX)= 0ABC_{16} + 1100_{16}=1BBC_{16}$$

نتیجه در AX ریخته میشود که هست :

$$(AX) = 1BBC_{16}$$

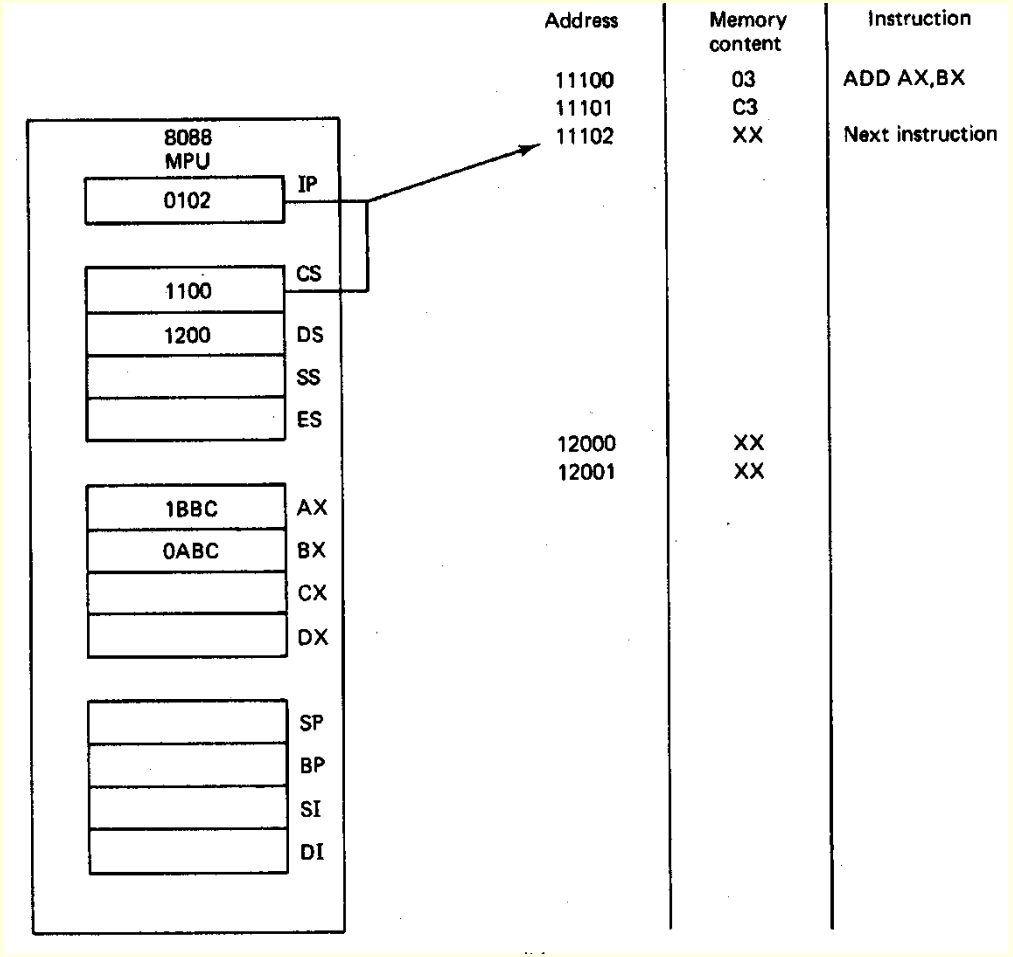
دستورالعمل های محاسباتی

- ADD AX, BX



دستورالعمل های محاسباتی

- ADD AX, BX



دستورالعمل های محاسباتی

- *مثال:*
- مقادیر اولیه AX ، bl، محتویات حافظه SUM و CARRY FLAG به ترتیب برابر با 1234_{16} ، AB_{16} ، $00CD_{16}$ و 0_{16} میباشند نتیجه اجرای دستورالعمل های زیر را توضیح دهید

```
ADD AX, [SUM]
ADC BL, 05H
INC WORD PTR [SUM]
```

- *راه حل:*

$(AX) \leftarrow (AX)+(SUM) = 1234_{16} + 00CD_{16} = 1301_{16}$

$(BL) \leftarrow (BL)+imm8+(CF) = AB_{16} + 5_{16}+0_{16} = B0_{16}$

$(SUM) \leftarrow (SUM)+ 1_{16} = 00CD_{16} + 1_{16} = 00CE_{16}$

AAA Instruction

- ASCII Adjust after Addition
- Corrects result in AH and AL after addition when working with BCD values.

- Example:
  - **MOV AX, 15** ; AH = 00, AL = 0Fh
  - **AAA** ; AH = 01, AL = 05
  - **RET**

DAA

- Decimal adjust After Addition.
- Corrects the result of addition of two packed BCD values.
- Example:
  - **MOV AL, 0Fh** ; AL = 0Fh (15)
  - **DAA** ; AL = 15h
  - **RET**

دستورالعمل های محاسباتی

- *مثال:*
- جمع 32 بیتی باینری را بر روی محتوای رجیستر های پروسسور انجام دهید

- *راه حل:*

(DX,CX) ⟵ (DX,CX) + (BX,AX)

(DX,CX) = $FEDCBA98_{16}$

(BX,AX) = 01234567_{16}

```
MOV DX, 0FEDCH
MOV CX, 0BA98H
MOV BX, 01234H
MOV AX, 04567H
ADD CX, AX
ADC DX, BX ;
```

دستورالعمل های محاسباتی

- دستورالعمل های تفریق: SUB, SBB
- NEG , DEC, AAS, DAS,

mnemonic	مفهوم	format	OPERATION	تاثیر بر فلگ ها
sub	تفریق کن	Sub d,s	(d)-(s)→(d) Borrow→(cf)	OF,SF,ZF,AF,PF,CF
sbb	با کری تفریق کن	Sbb d,s	(D)-(S)-(CF)→(D)	OF,SF,ZF,AF,PF,CF
dec	یکی کم کن	Dec d	(D)-1→(D)	OF,SF,ZF,AF,PF
neg	منفی کن	Neg d	0-(D)→(D) 1→(CF)	OF,SF,ZF,AF,PF,CF
das	تنظیم اسکی برای تفریق	das		SF,ZF,AF,PF,CF,OF, تعریف نشده
aas	تنظیم ده تایی برای تفریق	aas		AF,CF,OF,SF,ZF,PF تعریف نشده

دستورالعمل های محاسباتی

- دستورالعمل های تفریق: NEG SUB, SBB, DEC, AAS, DAS,

مقصد	مبدا
رجیستر	رجیستر
رجیستر	حافظه
حافظه	رجیستر
رجیستر	عدد ناگهانی
جافظه	عدد ناگهانی

عملگرهای مجاز برای SUBو SBB

مقصد
رجیستر 16 تایی
رجیستر 8 تایی
حافظه

عملگرهای مجاز برای DEC

مقصد
رجیستر
حافظه

عملگرهای مجاز برای NEG

دستورالعمل های محاسباتی

- *مثال:*
- در نظر بگیرید که محتوای رجیسترهای BX و CX به ترتیب 1234_{16} و 0123_{16} هستندو مقدار CARRY FLAG صفر است. نتیجه اجرای دستورالعمل SBB BX, CX چیست ؟

- *راه حل:*

$$(BX)\text{-}(CX)\text{-}(CF) \longrightarrow (BX)$$

پس داریم:

$$(BX) = 1234_{16} - 0123_{16} - 0_{16}$$
$$= 1111_{16}$$

CARRY FLAG صفر باقی میماند.

دستورالعمل های محاسباتی

- *مثال:*
- در نظر بگیرید که رجیستر BX دارای مقدار $003A_{16}$ می باشد. نتیجه اجرای دسترالعمل زیر چیست؟

NEG BX

- *راه حل:*

$(BX) = 0000_{16}-(BX)=0000_{16}+2'complement\ of \quad 003A_{16}$

$= 0000_{16}+FFC6_{16}$

$= FFC6_{16}$

چون هیچ CARRY در دستورالعمل جمع ایجاد نمیشود پس مقدار CARRY FLAG به مقدار متمم آن یعنی CF=1 میشود

دستورالعمل های محاسباتی

- *مثال*
- تفریق 32 بیتی را برای متغیر های X و Y بنویسید
- *راه حل:*

MOV	SI,200H	معرفی نشانگر برای X
MOV	DI,100H	معرفی نشانگر برای Y
MOV	AX,[SI]	ورد های LS را تفریق کن
SUB	AX,[DI]	
MOV	[SI],AX	ورد LS نتیجه را ذخیره کن
MOV	AX,[SI]+2	ورد MS را تفریق کن
SBB	AX,[DI]+2	
MOV	[SI]+2,AX	ورد MS از نتیجه را ذخیره کن

DAS

- Decimal adjust After Subtraction.
- Corrects the result of subtraction of two packed BCD values.
- Example:
  - **MOV AL, 0FFh** ; AL = 0FFh (-1)
  - **DAS** ; AL = 99h, CF = 1
  - **RET**

دستورالعمل های ضرب و تقسیم

MNEMONIC	مفهوم	FORMAT	OPERATION	تاثیر بر فلگ ها
MUL	ضرب بدون علامت	MUL S	(AL).(S8)→(AX) (AX).(S16)→(DX),(AX)	OF,CF,SF,ZF,AF,PF UNDEFINED
DIV	تقسیم بدون علامت	DIV S	(1) Q((AX)/(S8))→(AL) R((AX)/(S8))→(AH) (2) Q((DX,AX)/(S16))→(AX) R((DX,AX)/(S16))→(DX)	OF,SF,ZF,AF,PF,CF UNDEFINED
IMUL	ضرب علامت دار	IMUL S	(AL).(S8)→(AX) (AX).(S16)→(DX),(AX)	OF,CF,SF,ZF,AF,PF UNDEFINED
IDIV	تقسیم علامت دار	IDIV S	(1) Q((AX)/(S8))→(AL) R((AX)/(S8))→(AH) (2) Q((DX,AX)/(S16))→(AX) R((DX,AX)/(S16))→(DX)	OF,SF,ZF,AF,PF,CF UNDEFINED

MUL / IMUL

- MUL REG | memory
- when operand is a **byte**:
  AX = AL * operand.
- when operand is a **word**:
  (DX AX) = AX * operand.
- Example:
  - **MOV AL, 200** ; AL = 0C8h
  - **MOV BL, 4**
  - **MUL BL** ; AX = 0320h (800)
  - **RET**

DIV / IDIV

- DIV REG | MEM
  - when operand is a **byte**:
    AL = AX / operand
    AH = remainder (modulus)
  - when operand is a **word**:
    AX = (DX AX) / operand
    DX = remainder (modulus)
- Example:
  - **MOV AX, 203** ; AX = 00CBh
  - **MOV BL, 4**
  - **DIV BL** ; AL = 50 (32h), AH = 3
  - **RET**

دستورالعمل های محاسباتی

- *مثال:*
- محتوای AL برابر با عدد FF و CL محتوی عدد FE می باشد نتیجه حاصل در AX پس از اجرای دستورالعمل های زیر چیست ؟

MUL CL and IMUL CL

- *راه حل:*

(AL) = -1 (as 2's complement) = $11111111_2 = FF_{16}$

(CL) = -2 (as 2's complement) = $11111110_2 = FE_{16}$

پس از اجرای دستورالعمل MUL :

(AX) = $11111111_2 x 11111110_2 = 1111110100000010_2 = FD02_{16}$

پس از اجرای دستورالعمل IMUL :

(AX) = $-1_{16} \times -2_{16} = 2_{16} = 0002_{16}$

دستورالعمل های محاسباتی

AAM	تنظیم AL برای ضرب	AAM	Q((AL)/10)→(AH) R((AL)/10)→(AL)	SF, ZF, PF و OF, AF, CF نامعلوم است
AAD	تنظیم AX برای ضرب	AAD	(AH).10+(AL)→(AL) 00→(AH)	SF, ZF, PF و OF, AF, CF نامعلوم است
CBW	تبدیل بایت به ورد	CBW	(MSB OF AL)→(ALL BITS OF AH)	هیچ
CWD	تبدیل ورد به دابل ورد	CWD	(MSB OF AL)→(ALL BITS OF DX)	هیچ

مبدا
رجیستر 8 تایی
رجیستر 16تایی
حافظه 8 تایی
حافظه 16 تایی

AAD

- ASCII Adjust before Division.
  Prepares two BCD values for division.

  Algorithm:

  AL = (AH * 10) + AL, AH = 0

  Example:

```
MOV AX, 0105h ; AH = 01, AL = 05
AAD ; AH = 00, AL = 0Fh (15)
RET
```

دستورالعمل های محاسباتی

- *مثال:*
  - نتیجه اجرای دستورالعمل زیر چیست؟

```
MOV AL, 0A1H
CBW
CWD
```

- *راه حل:*

$(AL) = A1_{16} = 10100001_2$

اجرای دستورالعمل CBW مقدار MSB در ALرا تعمیم میدهد:

$(AH) = 11111111_2 = FF_{16}$

or $(AX) = 1111111110100001_2$

پس از اجرای CWD داریم:

$(DX) = 1111111111111111_2 = FFFF_{16}$

$(AX) = FFA1_{16}$ $(DX) = FFFF_{16}$

دستورالعمل های منطقی

- دستورالعملهای منطقی شامل موارد زیر می شوند:
  - AND
  - OR
  - XOR (Exclusive-OR)
  - NOT

کد	معنی	قالب	عملکرد	تاثیر بر FLAG
AND	AND منطقی	AND D,S	(S).(D) → (D)	OF, SF, ZF, PF, CF AF تاثیری ندارد
OR	OR منطقی	OR D,S	(S)+(D) →(D)	OF, SF, ZF, PF, CF AF تاثیری ندارد
XOR	XOR منطقی	XOR D,S	(S) ⊕ (D) →(D)	OF, SF, ZF, PF, CF AF تاثیری ندارد
NOT	NOT منطقی	NOT D	(%D) →(D)	هیچ

دستورالعمل های منطقی

- دستورالعملهای منطقی:AND, OR, XOR, NOT

مبدا	مقصد
رجیستر	رجیستر
حافظه	رجیستر
رجیستر	حافظه
عدد ناگهانی	رجیستر
عدد ناگهانی	حافظه

عملگرهای مجاز برای AND , OR , XOR

مقصد
رجیستر
حافظه

عملگرهای مجاز برای NOT

دستورالعمل های منطقی

- *مثال:*
  - نتیجه اجرای دستورالعمل های زیر را شرح دهید

```
MOV AL, 01010101B
AND AL, 00011111B
OR  AL, 11000000B
XOR AL, 00001111B
NOT AL
```

- *راه حل:*

$(AL)=01010101_2 \cdot 00011111_2 = 00010101_2 = 15_{16}$

پس از اجرای دستورالعمل OR داریم:

$(AL)= 00010101_2 + 11000000_2 = 11010101_2 = D5_{16}$

پس از اجرای دستورالعمل XOR داریم:

$(AL)= 11010101_2 \oplus 00001111_2 = 11011010_2 = DA_{16}$

پس از اجرای دستورالعمل NOT داریم:

$(AL)= (NOT)11011010_2 = 00100101_2 = 25_{16}$

دستورالعمل های منطقی

- *مثال:*
  - صفر کردن یا یک کردن بیت ها در رجیستر
- *راه حل:*

12 بیت بالایی ورد واقع در AXرا آشکار کنید

AND AX, $000F_{16}$

- بیت B_4 از آدرس OFFSET بایت CONTROL-FLAG را یک کنید

```
MOV   AL, [CONTROL_FLAGS]
OR    AL, 10H
MOV   [CONTROL_FLAGS], AL
```

پس از اجرای دستورالعمل بالا داریم:

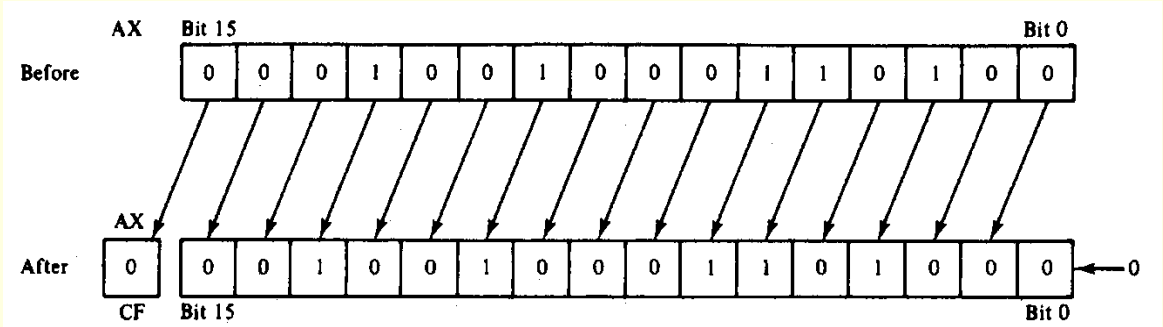
$(AL)=XXXXXXXX_2 + 00010000_2 = XXX1XXXX_2$

SAR

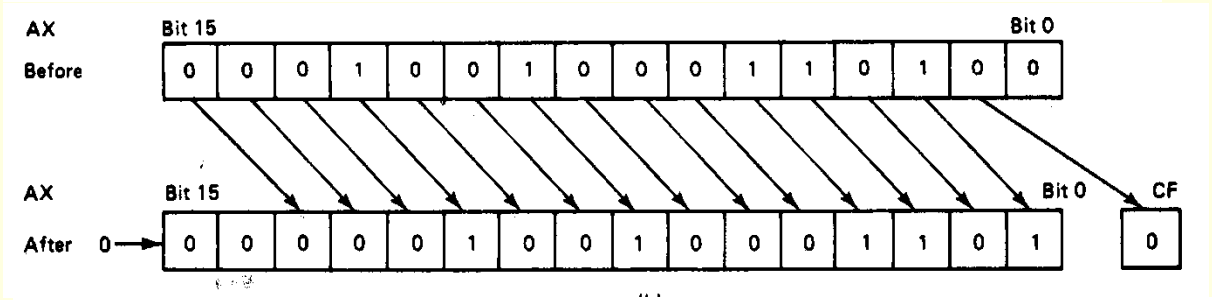
- SAR memory, immediate | REG, immediate | memory, CL | REG, CL
- Shift Arithmetic operand1 Right. The number of shifts is set by operand2.
  Example:
  - **MOV AL, 0E0h** ; AL = 11100000b
  - **SAR AL, 1** ; AL = 11110000b, CF=0.
  - **MOV BL, 4Ch** ; BL = 01001100b
  - **SAR BL, 1** ; BL = 00100110b, CF=0.
  - **RET**

دستورالعمل های شیفت دادن

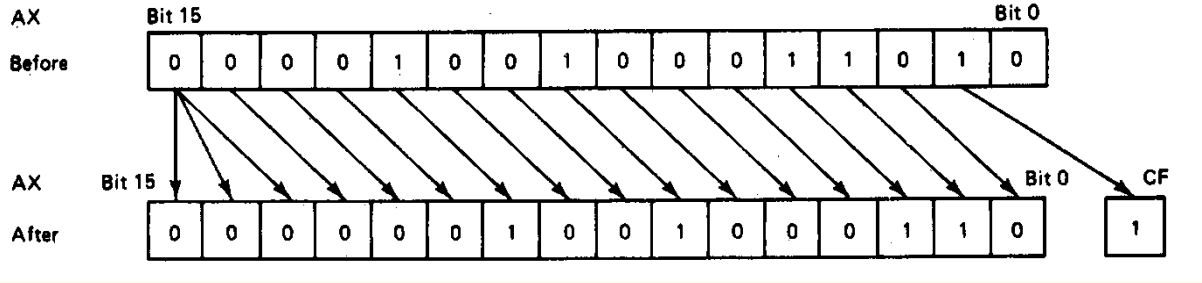
- دستورالعملهای شیفت دادن: SHL, SHR, SAL, SAR



SHL AX, 1



SHR AX, CL
(CL)=2



SAR AX, CL
(CL)=2

دستورالعمل های شیفت دادن

- *مثال:*
- در نظر بگیرید که CL درای مقدار 02_{16} و AX دارای مقدار $091A_{16}$ می باشد. تعیین کنید که مقادیر جدید AX و CARRY FLAG پس از اجرای دستورالعمل SAR AX, CL چیست؟

- *راه حل:*

$(AX)=0000001001000110_2=0246_{16}$

carry flag : $(CF)=1_2$

دستورالعمل های شیفت دادن

- *مثال:*
  - مقدار بیت B3 از محتویات آدرس OFFSET زیر در حافظه را جدا کنید
  - CONTROL_FLAGS.
- *راه حل*

```
MOV AL, [CONTROL_FLAGS]
MOV CL, 04H
SHR AL, CL
```

پس از اجرای این دستورالعمل ها داریم:

$(AL)=0000B_7B_6B_5B_4$ and $(CF)=B_3$

دستورالعمل های چرخشی

- دستورالعمل های چرخشی: ROL, ROR, RCL, RCR

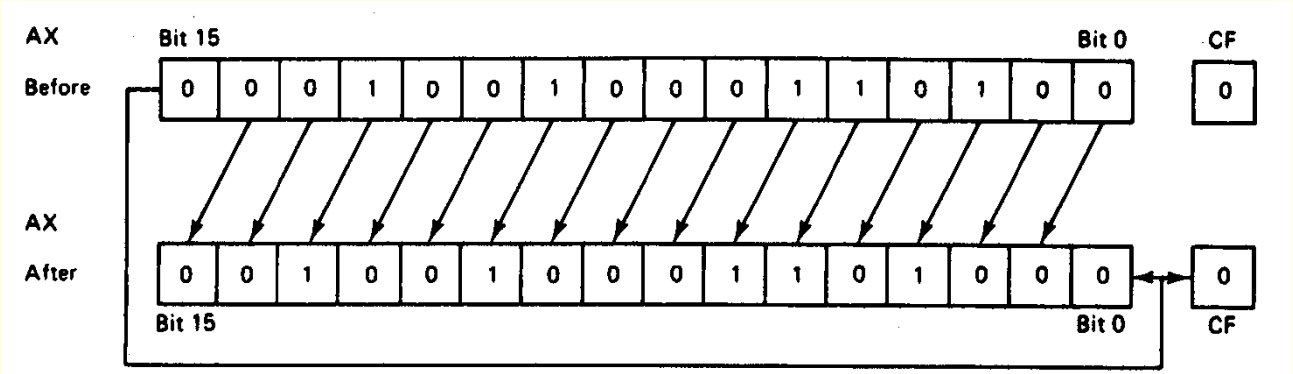
کد	معنی	قالب	عملکرد	تاثیر بر flag
ROL	به چپ بچرخان	ROL D,COUNT	تعیین Count را به اندازه عددی که در D شده به سمت چپ بچرخان هربیتی که از سمت چپ به بیرون شیفت پیدا میکند به سمت راستی ترین بیت برمیگردد.	CF OF نامعلوم است اگر COUNT برابر یک نباشد
ROR	به راست بچرخان	ROR D,COUNT	تعیین Count را به اندازه عددی که در D شده به سمت راست بچرخان هربیتی که از سمت راست به بیرون شیفت پیدا میکند به سمت چپی ترین بیت برمیگردد	CF OF نامعلوم است اگر COUNT برابر یک نباشد
RCL	به چپ بچرخان با کری	RCL D, COUNT	است به جز اینکه کری برای ROLمانند ملحق میشود.Dچرخش به	CF OF نامعلوم است اگر COUNT یک نباشد
RCR	به راست بچرخان با کری	RCR D,COUNT	است به جز اینکه کری برای RORمانند ملحق میشود.Dچرخش به	CF OF نامعلوم است اگر COUNT یک نباشد

ROR

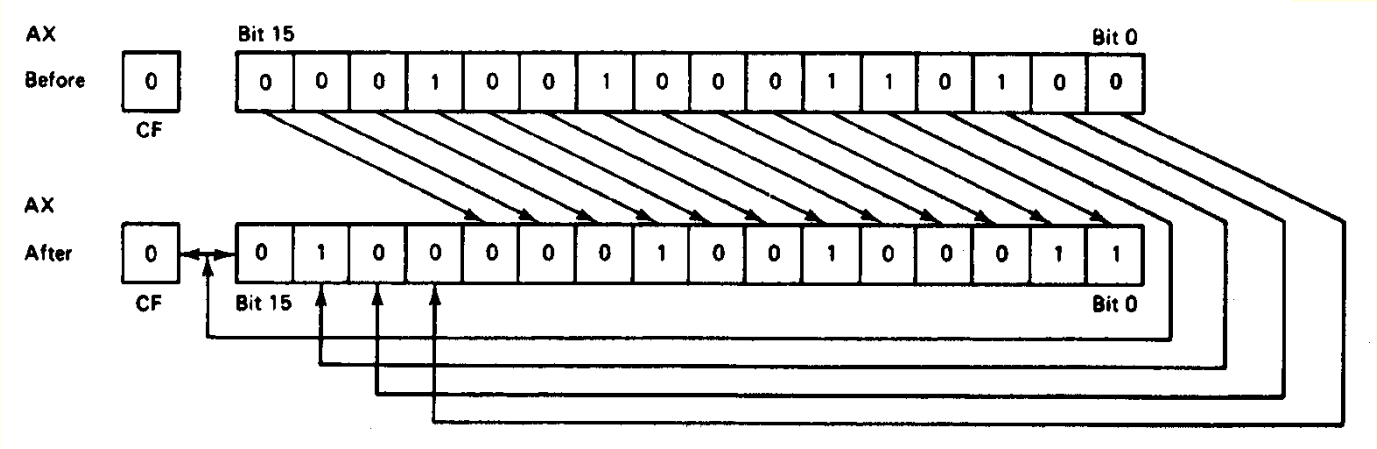
- ROR memory, immediate | REG, immediate | memory, CL | REG, CL
- Rotate operand1 right. The number of rotates is set by operand2.
- Algorithm:
  shift all bits right, the bit that goes off is set to CF and the same bit is inserted to the left-most position.
- Example:
  - **MOV AL, 1Ch** ; AL = 00011100b
  - **ROR AL, 1** ; AL = 00001110b, CF=0.
  - **RET**

دستورالعمل های چرخشی

- دستورالعمل های چرخشی: ROL, ROR, RCL, RCR



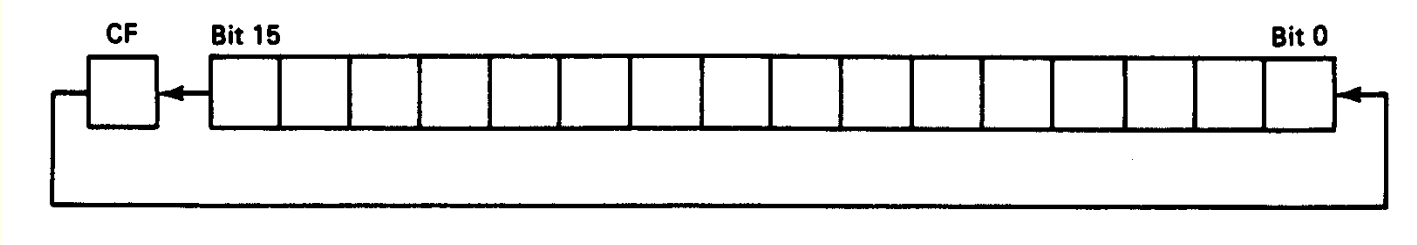
ROL AX, 1



ROR AX, CL
(CL)=4

RCR & RCL

برای دستورالعمل های RCL و RCL بیت ها به همراه CARRY FLAG میچرخند



دستورالعمل های چرخشی

- *مثال:*
- نتیجه حاصله در BX و CF پس از اجرای دستورالعمل زیر چیست؟

RCR BX, CL

فرض کنید که مقادیر اولیه قبل از اجرای دستورالعمل مقادیر زیر هستند:

$(CL)=04_{16}$, $(BX)=1234_{16}$, and $(CF)=0$

- *راه حل:*

مقدار اولیه BXهست:

$$(BX) = 0001001000110100_2 = 1234_{16}$$

- اجرای دستور RCR موجب چرخش چهار بیتی به سمت راست به همراه کری می شود که نتیجه حاصل در BX برابر است با:

$$(BX) = 1000000100100011_2 = 8123_{16}$$
$$(CF) = 0_2$$

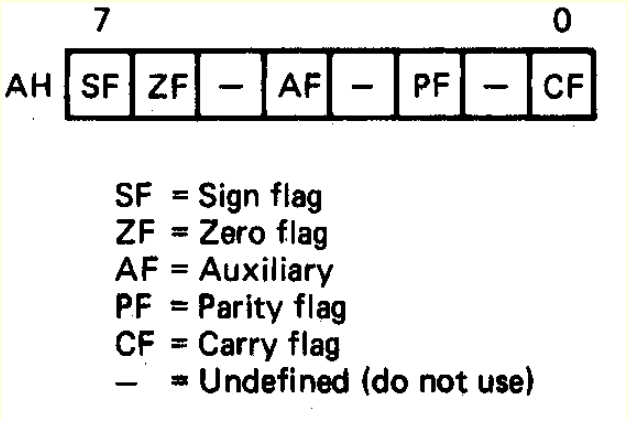
دستورالعمل های کنترل FLAG

- دستورالعمل هایی که برای کنترل کردن FLAG هستند و زمانی که اجرا میشوند مستثیما حالت FLAG را تحت تاثیر قرار می دهند

- LAHF (مقادیر FLAG را در AH بارگذاری کن)
- SAHF (مقدار AH را در FLAG ذخیره کن)
- CLC(کری را پاک کن)
- STC(کری را یک کن)
- CMC(کری را متمم کن)
- CLI(وقفه را پاک کن)
- STI(وقفه را یک کن)

دستورالعمل های کنترل FLAG

کد	معنی	عملکرد	تاثیر بر FLAG
LAHF	FLAG را در AH بارگذاری کن	(AH) ← (FLAGS)	هیچ
SAHF	AH را در FLAG ذخیره کن	(FLAGS) ← (AH)	SF, ZF, AF, PF, CF
CLC	CF را صفر کن	(CF) ←0	CF
STC	CF را یک کن	(CF) ←1	CF
CMC	CF را متمم کن	(CF) ←(%CF)	CF
CLI	IF (وقفه FLAG) را صفر کن	(IF) ←0	IF
STI	IF (وقفه FLAG) را یک کن	(IF) ←1	IF



دستورالعمل های کنترل FLAG

- *مثال:*
- دستورالعملی بنویسید که مقادیر حاضر FLAG را در مکانی از حافظه واقع در OFFSET متغیر MEM1 ذخیره کند و سپس مقادیر FLAG را با مقدار ذخیره شده در خانه حافظه واقع در OFFSET متغیر MEM2 بارگذاری کند
- *راه حل:*

LAHF	مقدار FLAG را در AHبارگذاری کن
MOV [MEM1], AH	مقدار AHرا در MEM1بریز
MOV AH, [MEM2]	مقدار MEM2 را در AH بارگذاری کن
SAHF	محتوای AHرا در FLAG ذخیره کن

دستورالعمل های مقایسه ای

- دستورالعمل های مقایسه ای:

- دستورالعمل های مقایسه ای ما را قادر می سازد که رابطه بین دو عدد را تعیین کنیم

کد	معنی	قالب	عملکرد	تاثیر بر FLAG
CMP	مقایسه	CMP D,S	برای تعیین و صفر و یک کردن FLAG ها از D-S استفاده میکند	CF, AF, OF, PF, SF, ZF

مبدا	مقصد
رجیستر	رجیستر
حافظه	رجیستر
رجیستر	حافظه
مقدار ناگهانی	رجیستر
مقدار ناگهانی	حافظه
مقدار ناگهانی	مقدار ناگهانی

Flags After CMP

Condition Code Settings After CMP

Unsigned operands:	Signed operands:
Z: equality/inequality	Z: equality/inequality
C: Oprnd1 < Oprnd2 (C=1) Oprnd1 >= Oprnd2 (C=0)	C: no meaning
S: no meaning	S: see below
O: no meaning	O: see below

For signed comparisons, the S (sign) and O (overflow) flags, taken together, have the following meaning:

If ((S=0) and (O=1)) or ((S=1) and (O=0)) then Oprnd1 < Oprnd2 when using a signed comparison.

If ((S=0) and (O=0)) or ((S=1) and (O=1)) then Oprnd1 >= Oprnd2 when using a signed comparison.

To understand why these flags are set in this manner, consider the following examples:

Oprnd1	minus	Oprnd2	S	O
------		------	-	-
OFFFF (-1)	-	OFFFE (-2)	0	0
08000	-	00001	0	1
OFFFE (-2)	-	OFFFF (-1)	1	0
07FFF (32767)	-	OFFFF (-1)	1	1

دستورالعمل های مقایسه ای

- مثال:
- شرح دهید پس از اجرای دستورالعملهای زیر FLAG ها در چه وضعیتی هستند

```
MOV AX, 1234H
MOV BX, 0ABCDH
CMP AX, BX
```

- راه حل:

$(AX) = 1234_{16} = 0001001000110100_2$

$(BX) = ABCD_{16} = 1010101111001101_2$

$(AX) – (BX) = 0001001000110100_2 – 1010101111001101_2 = 0110011001100111_2$

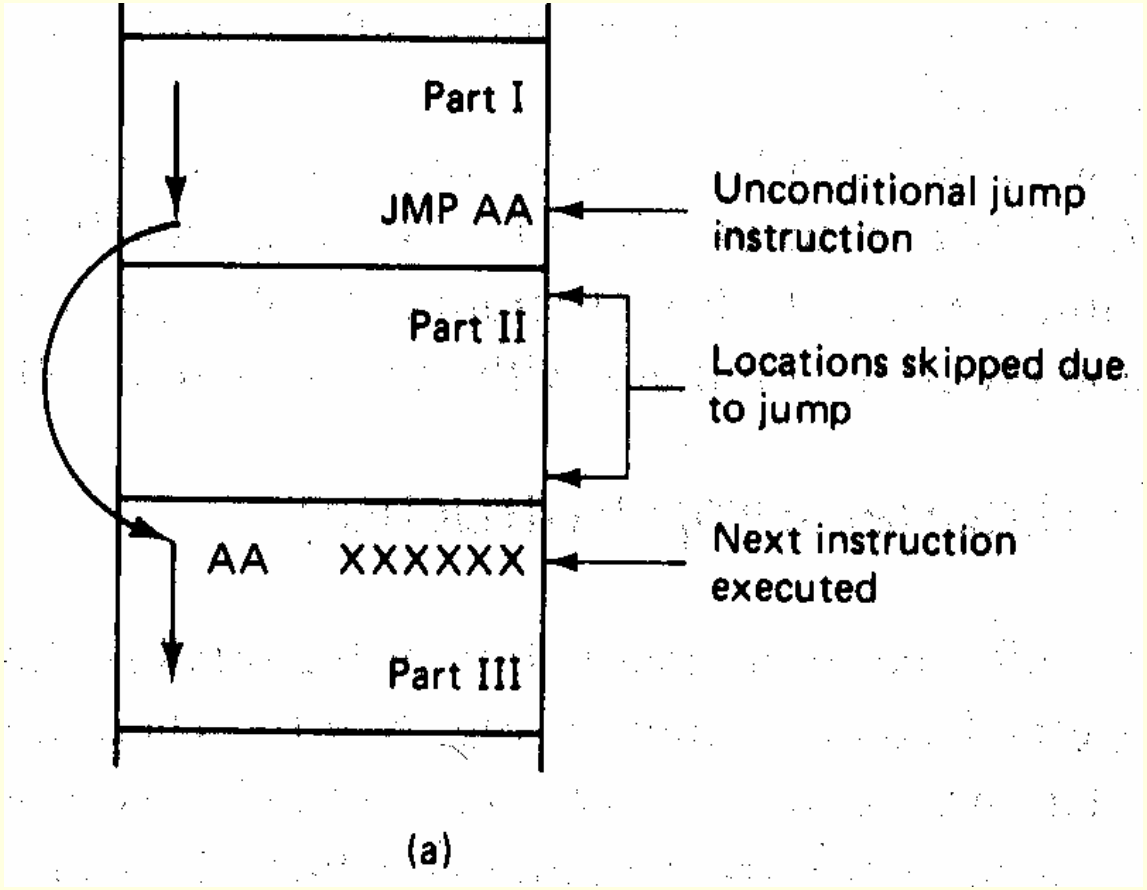
بنابراین AF = 1, CF = 1, PF = 0, OF = 0, SF = 0, ZF = 0 ,

دستورالعمل های کنترل روند و جهشی

- دستور جهش غیر شرطی
- دستور جهش شرطی
- دستورالعمهای انشعاب IF-THEN
- دستورالعمل های حلقه: REPEAT-UNTIL و WHILE-DO
- برنامه هایی که از دستورالعمل های حلقه و انشعاب استفاده میکنند

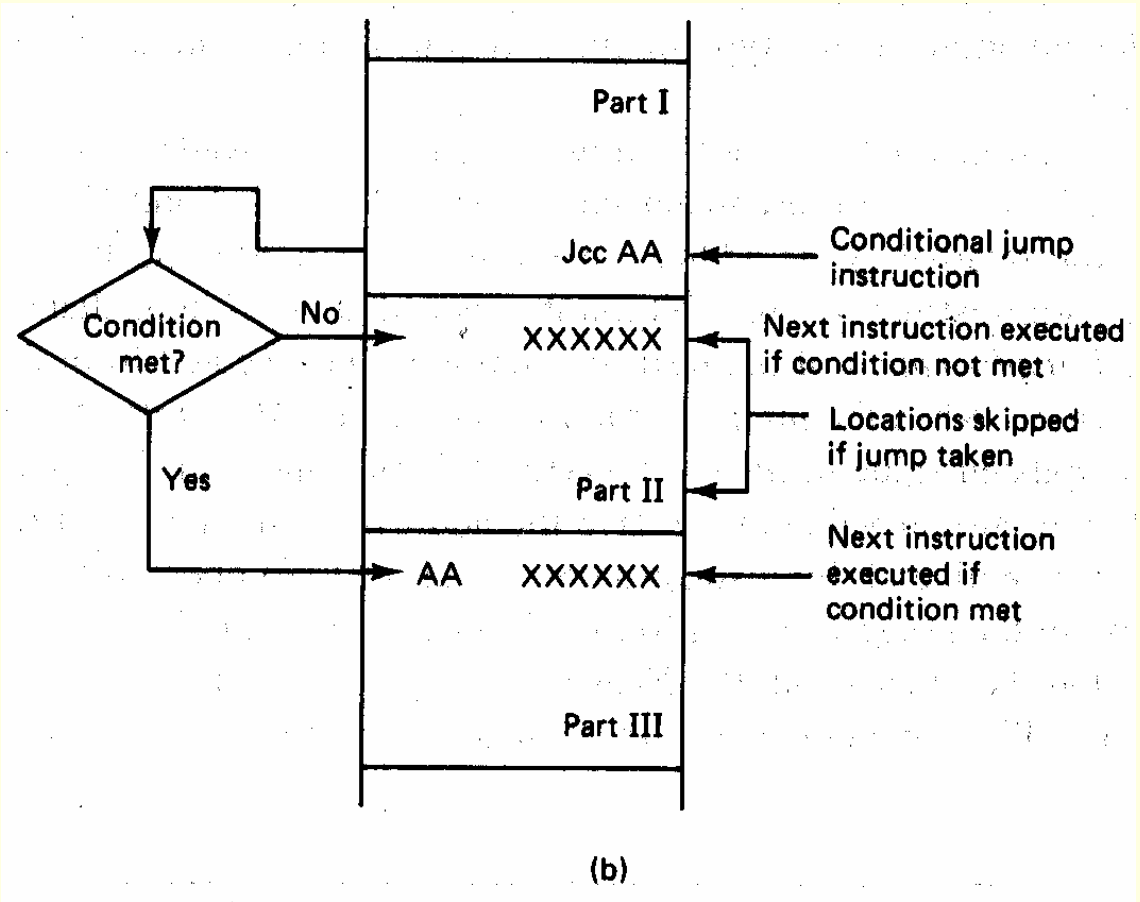
دستورالعمل های کنترل روند و جهشی

- جهشهای شرطی و غیر شرطی



دستورالعمل های کنترل روند و جهشی

- جهشهای شرطی و غیر شرطی



دستورالعمل های کنترل روند و جهشی

- دستورالعمل جهش غیر شرطی:

کد	معنی	قالب	عملکرد	تاثیر بر FLAG
JMP	جهش غیر شرطی	JMP OPERAND	جهش از آدرسی که توسط عملوند تعیین شده شروع میشود	هیچ

عملوند
LABEL کوتاه
LABEL نزدیک
LABEL دور
حافظه 16 تایی
رجیستر 16 تایی
حافظه 32 تایی

دستورالعمل های کنترل روند و جهشی

- *مثال*
- به BX جهش کنید در حالیکه BX محتوی مقدار 0010_{16} می باشد
- JMP [BX].

دستورالعمل های کنترل روند و جهشی

کد	معنی	قالب	عملکرد	تاثیر بر FLAG
JCC	جهش شرطی	JCC OPERAND	اگر شرایط تعیین شده CC صادق باشد جهش از آدرسی که توسط عملوند تعیین شده شروع میشود درغیراین صورت دستورالعمل بعدی اجرا میشود.	هیچ

دستورالعمل های کنترل روند و جهشی

دستورالعمل های جهش شرطی

کد	معنی	شرایط
JA	بالاتر	CF=0 , ZF=0
JAE	بالاتر یا مساوی	CF=0
JB	پایین تر	CF=1
JBE	پایین تر یا مساوی	CF=1 یا ZF=1
JC	کری	CF=1
JCXZ	رجیستر CX برابر صفر است	(CF یا ZF)=0
JE	برابر	ZF=1
JG	بزرگتر	ZF=0 و SF=OF
JGE	بزرگتر یا مساوی	SF=OF
JL	کمتر	(SF یا OF) =1
JLE	کمتر یا مساوی	((SF یا OF) یا ZF)=1
JNA	بالاتر نیست	CF=1 یا ZF=1
JNAE	نه بالاتر است نه مساوی	CF=1
JNB	پایین تر نیست	CF=0
JNBE	نه پایین تر است نه مساوی	CF=0 و ZF=0

دستورالعمل های کنترل روند و جهشی

دستورالعمل های جهش شرطی

JNC	کری نیست	CF = 0
JNE	برابر نیست	ZF = 0
JNG	بزرگتر نیست	((SFیا OF) یا ZF)=1
JNGE	نه بزرگتر است نه مساوی	(SFیا OF)= 1
JNL	کوچکتر نیست	SF= OF
JNLE	نه کوچکتر است نه مساوی	ZF=0 و SF = OF
JNO	سرریز نیست	OF=0
JNP	نیست PARITY	PF=0
JNS	علامت دار نیست	SF=0
JNZ	صفر نیست	ZF=0
JO	سرریز شده	OF=1
JP	هست PARITY	PF
JPE	زوج است PARITY	1
JPO	فرد است PARITY	PF=1
JS	علامت دار است	SF=1
JZ	صفر است	ZF=1

Conditional Jump

Instruction	Description	signed-ness	Flags	short jump opcodes	near jump opcodes
JO	Jump if overflow		OF = 1	70	0F 80
JNO	Jump if not overflow		OF = 0	71	0F 81
JS	Jump if sign		SF = 1	78	0F 88
JNS	Jump if not sign		SF = 0	79	0F 89
JE **JZ**	Jump if equal Jump if zero		ZF = 1	74	0F 84
JNE **JNZ**	Jump if not equal Jump if not zero		ZF = 0	75	0F 85
JB **JNAE** **JC**	Jump if below Jump if not above or equal Jump if carry	unsigned	CF = 1	72	0F 82
JNB **JAE** **JNC**	Jump if not below Jump if above or equal Jump if not carry	unsigned	CF = 0	73	0F 83
JBE **JNA**	Jump if below or equal Jump if not above	unsigned	CF = 1 or ZF = 1	76	0F 86

Conditional Jump

Instruction	Description	signed-ness	Flags	short jump opcodes	near jump opcodes
JA **JNBE**	Jump if above Jump if not below or equal	unsigned	CF = 0 and ZF = 0	77	0F 87
JL **JNGE**	Jump if less Jump if not greater or equal	signed	SF <> OF	7C	0F 8C
JGE **JNL**	Jump if greater or equal Jump if not less	signed	SF = OF	7D	0F 8D
JLE **JNG**	Jump if less or equal Jump if not greater	signed	ZF = 1 or SF <> OF	7E	0F 8E
JG **JNLE**	Jump if greater Jump if not less or equal	signed	ZF = 0 and SF = OF	7F	0F 8F
JP **JPE**	Jump if parity Jump if parity even		PF = 1	7A	0F 8A
JNP **JPO**	Jump if not parity Jump if parity odd		PF = 0	7B	0F 8B

دستورالعمل های کنترل روند و جهشی

- ساختار برنامه انشعاب IF-THEN

```
        CMP   AX, BX
        JE    EQUAL
        ---   ---          ; Next instruction if (AX) ≠ (BX)
              .
              .
EQUAL:  ---   ---          ; Next instruction if (AX) = (BX)
              .
              .
        ---   ---
```

دستورالعمل های کنترل روند و جهشی

- ساختار برنامه انشعاب IF-THEN

```
                AND   AL, 04H
                JNZ   BIT2_ONE
                ---   ---          ; Next instruction if B2 of AL = 0
                      .
                      .
                ---   ---
BIT2_ONE:       ---   ---          ; Next instruction if B2 of AL = 1
                      .
                      .
                ---   ---
```

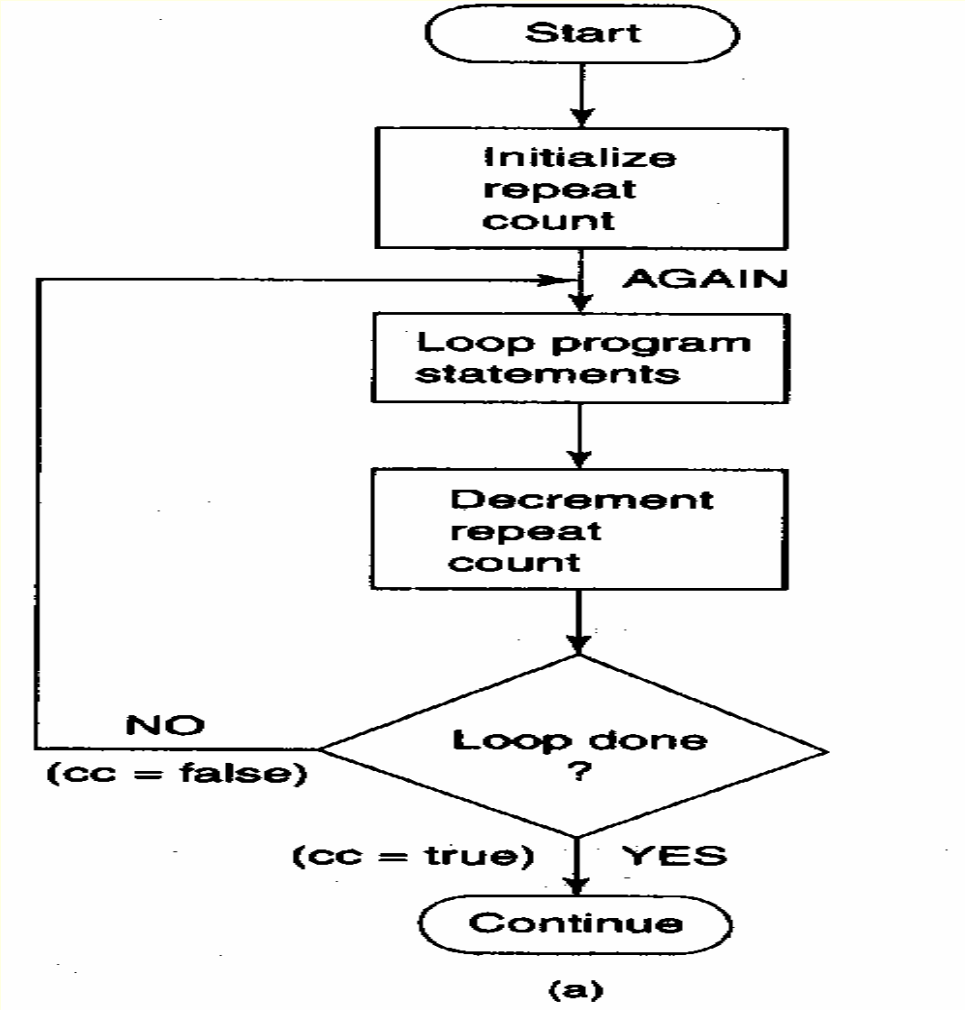
دستورالعمل های کنترل روند و جهشی

- ساختار برنامه انشعاب IF-THEN

```
                MOV   CL,03H
                SHR   AL, CL
                JC    BIT2_ONE
                ---   ---          ; Next instruction if B2 of AL = 0
                      .
                      .
                ---   ---
BIT2_ONE:       ---   ---          ; Next instruction if B2 of AL = 1
                      .
                      .
                ---   ---
```

دستورالعمل های کنترل روند و جهشی

- ساختار برنامه حلقه REPEAT-UNTIL



دستورالعمل های کنترل روند و جهشی

- ساختار برنامه حلقه REPEAT-UNTIL

```
        MOV   CL,COUNT
AGAIN:  ---   ---
        ---   ---
        .     .
        .     .
        .     .
        ---   ---
        DEC   CL
        JNZ   AGAIN
        ---   ---
```

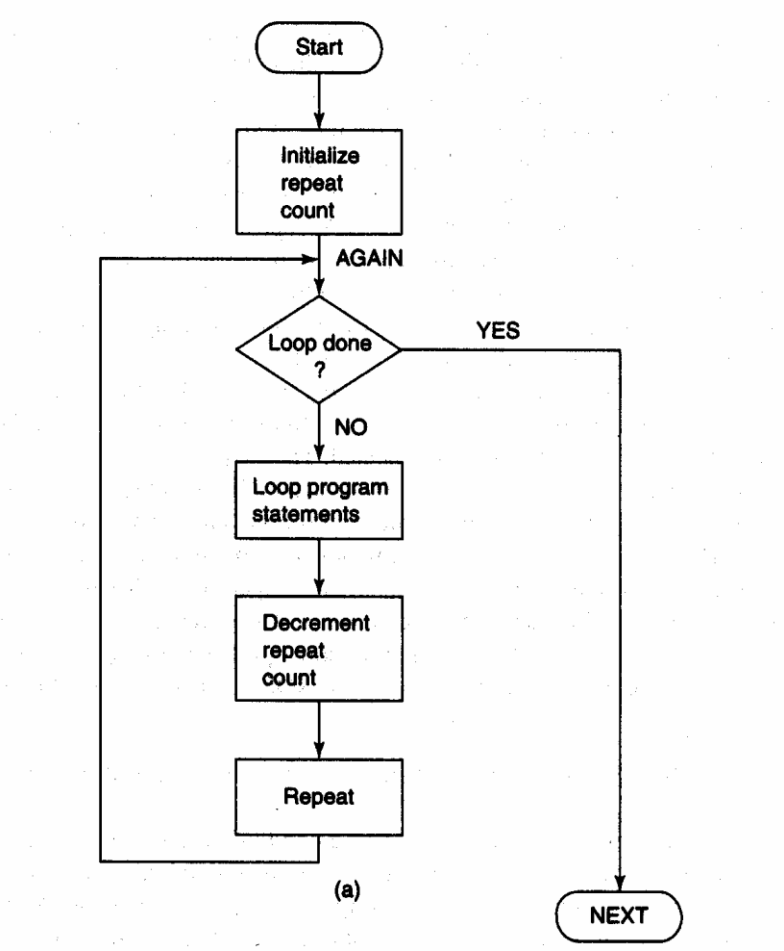
متغیر حلقه تکرار را تعیین کن
اولین دستورالعمل از حلقه
دومین دستورالعمل از حلقه

N امین دستورالعمل از حلقه
یکی از متغیر حلقه کم کن
اگر CL برابر صفر نیست یا ZF صفر نیست تکرار کن
اولین دستورالعمل بعد از خلقه اجرا میشود
کامل کن، CL=00H, ZF=1

معمول دستورالعمل REPEAT-UNTIL

دستورالعمل های کنترل روند و جهشی

- دستورالعمل برنامه حلقه WHILE-DO



WHILE-DO program sequence

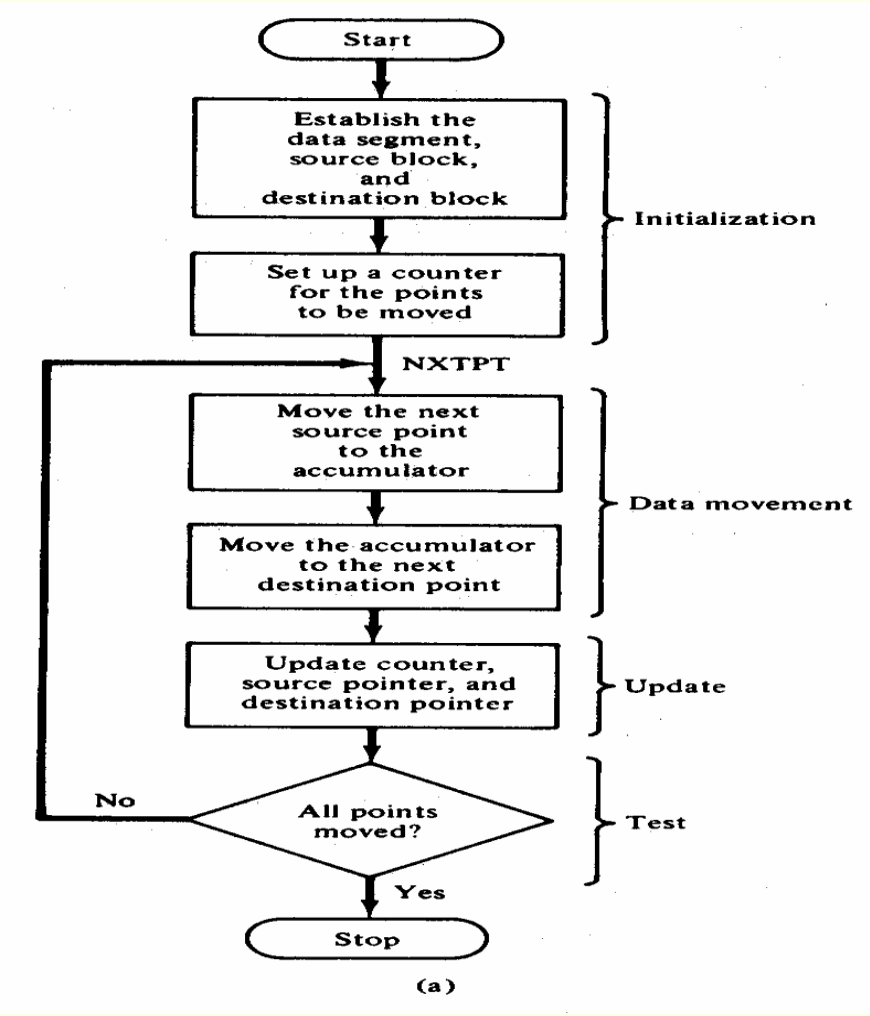
دستورالعمل های کنترل روند و جهشی

- دستورالعمل برنامه حلقه WHILE-DO

```
        MOV   CL,COUNT   ;Set loop repeat count
AGAIN:  JZ    NEXT       ;Loop is complete if CL = 00H (ZF = 1)
        ---   ---        ;1st instruction of loop
        ---   ---        ;2nd instruction of loop
         .     .               .
         .     .               .
         .     .               .
        ---   ---        ;nth instruction of loop
        DEC   CL         ;Decrement CL by 1
        JMP   AGAIN      ;Repeat from AGAIN
NEXT:   ---   ---        ;First instruction executed after loop is complete
```

(b)

دستورالعمل های کنترل روند و جهشی



- مثال – برنامه بلوک – حرکت (BLOCK-MOVE)

دستورالعمل های کنترل روند و جهشی

- مثال – برنامه بلوک – حرکت BLOCK-MOVE

```
                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
                DEC     CX
                JNZ     NXTPT
                HLT
```

(b)

Control Flow and Jump Instructions

- *EXAMPLE:*
  - Implement an instruction sequence that calculates the absolute difference between the contents of AX and BX and places it in DX.
- *Solution:*

```
        CMP AX, BX
        JNC DIFF2
DIFF1:  MOV DX, BX
        SUB DX, AX      ; (DX)=(BX)-(AX)
        JMP DONE
DIFF2:  MOV DX, AX
        SUB DX, BX      ; (DX)=(AX)-(BX)
DONE:   NOP
```

Instruction	Description	signed-ness	Flags	short jump opcodes	near jump opcodes
JO	Jump if overflow		OF = 1	70	0F 80
JNO	Jump if not overflow		OF = 0	71	0F 81
JS	Jump if sign		SF = 1	78	0F 88
JNS	Jump if not sign		SF = 0	79	0F 89
JE **JZ**	Jump if equal Jump if zero		ZF = 1	74	0F 84
JNE **JNZ**	Jump if not equal Jump if not zero		ZF = 0	75	0F 85
JB **JNAE** **JC**	Jump if below Jump if not above or equal Jump if carry	unsigned	CF = 1	72	0F 82
JNB **JAE** **JNC**	Jump if not below Jump if above or equal Jump if not carry	unsigned	CF = 0	73	0F 83
JBE **JNA**	Jump if below or equal Jump if not above	unsigned	CF = 1 or ZF = 1	76	0F 86

JA **JNBE**	Jump if above Jump if not below or equal	unsigned	CF = 0 and ZF = 0	77	0F 87
JL **JNGE**	Jump if less Jump if not greater or equal	signed	SF <> OF	7C	0F 8C
JGE **JNL**	Jump if greater or equal Jump if not less	signed	SF = OF	7D	0F 8D
JLE **JNG**	Jump if less or equal Jump if not greater	signed	ZF = 1 or SF <> OF	7E	0F 8E
JG **JNLE**	Jump if greater Jump if not less or equal	signed	ZF = 0 and SF = OF	7F	0F 8F
JP **JPE**	Jump if parity Jump if parity even		PF = 1	7A	0F 8A
JNP **JPO**	Jump if not parity Jump if parity odd		PF = 0	7B	0F 8B

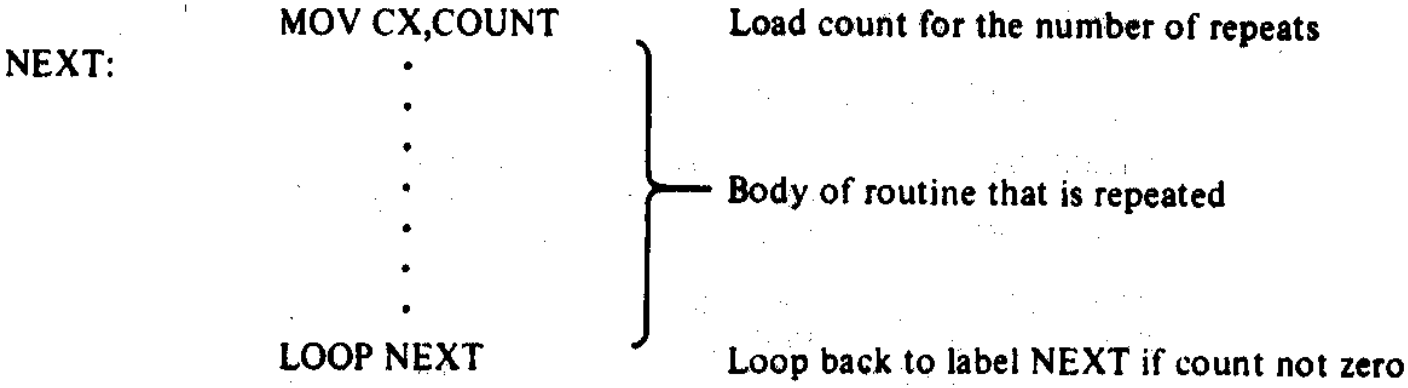
The Loop and the Loop-handling Instructions

- The LOOP 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

The Loop and the Loop-handling Instructions

- The LOOP instructions



The Loop and the Loop-handling Instructions

- Example – The block-move program

```
        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
```

The Loop and the Loop-handling Instructions

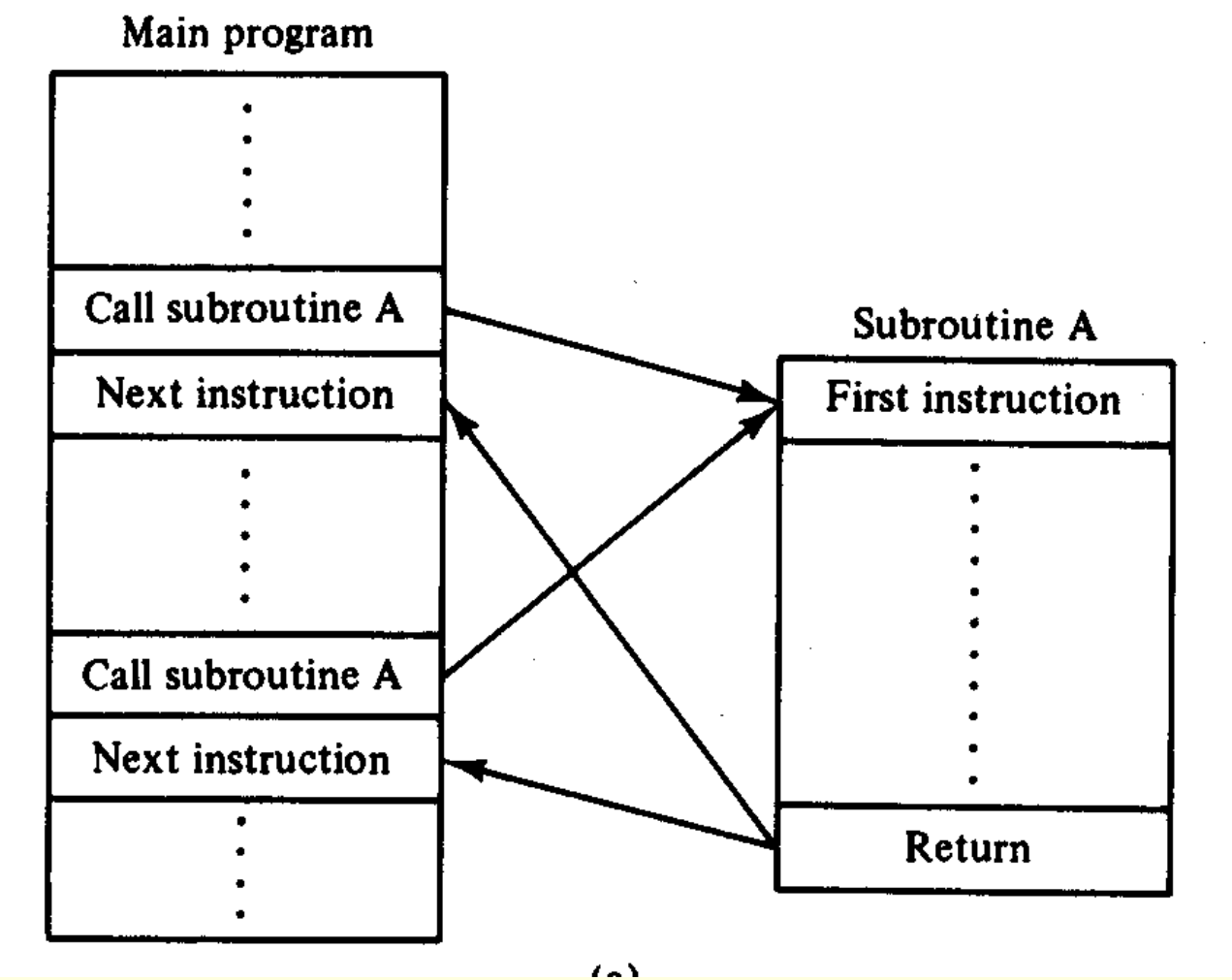
- *EXAMPLE:* Given the following sequence of instructions, explain what happens as they are executed

```
        MOV DL, 05
        MOV AX, 0A00H
        MOV DS, AX
        MOV SI, 0
        MOV CX, 0FH
AGAIN:  INC SI
        CMP [SI], DL
        LOOPNE AGAIN
```

Subroutines and Subroutine-Handling Instructions

- A subroutine is a special program that can be called for execution from any point in a program.
  - A subroutine is also known as a procedure.
- A return instruction must be included at the end of the subroutine to initiate the return sequence to the main program environment.
- CALL and RET instructions
- PUSH and POP instructions

Subroutines and Subroutine-Handling Instructions



Subroutines and Subroutine-Handling Instructions

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

(b)

Operand
Near-proc
Far-proc
Memptr16
Regptr16
Memptr32

(c)

Far Call & Near Call

- **The far call instruction does the following:**
  - It *pushes* the *cs* register onto the stack.
  - It pushes the *16 bit offset* of the next instruction following the call onto the stack.
  - It copies the 32 bit effective address into the *cs:ip* registers.
  - Execution continues at the *first instruction* of the subroutine.
- **The near call instruction does the following:**
  - It *pushes* the 16 bit offset of the next instruction following the call onto the stack.
  - It copies the 16 bit effective address into the *ip* register.
  - Execution continues at the *first instruction* of the subroutine.

Subroutines and Subroutine-Handling Instructions

Mnemonic	Meaning	Format	Operation	Flags Affected
RET	Return	RET or RET Operand	Return to the main program by restoring IP (and CS for far-proc). If Operand is present, it is added to the contents of SP.	None

(a)

Operand
None Disp16

(b)

RET Instruction

- RET No operands or even immediate Return from near procedure.

  Algorithm:

  Pop from stack: IP

  If immediate operand is present: SP = SP + operand

  Example:

```
#make_COM#
ORG 100h ; for COM file.
CALL  p1
ADD AX, 1
RET ; return to OS.

p1 PROC ; procedure declaration.
        MOV AX, 1234h
        RET ; return to caller.
p1 ENDP
```

RETF Instruction

RETF No operands or even immediate Return from Far procedure.

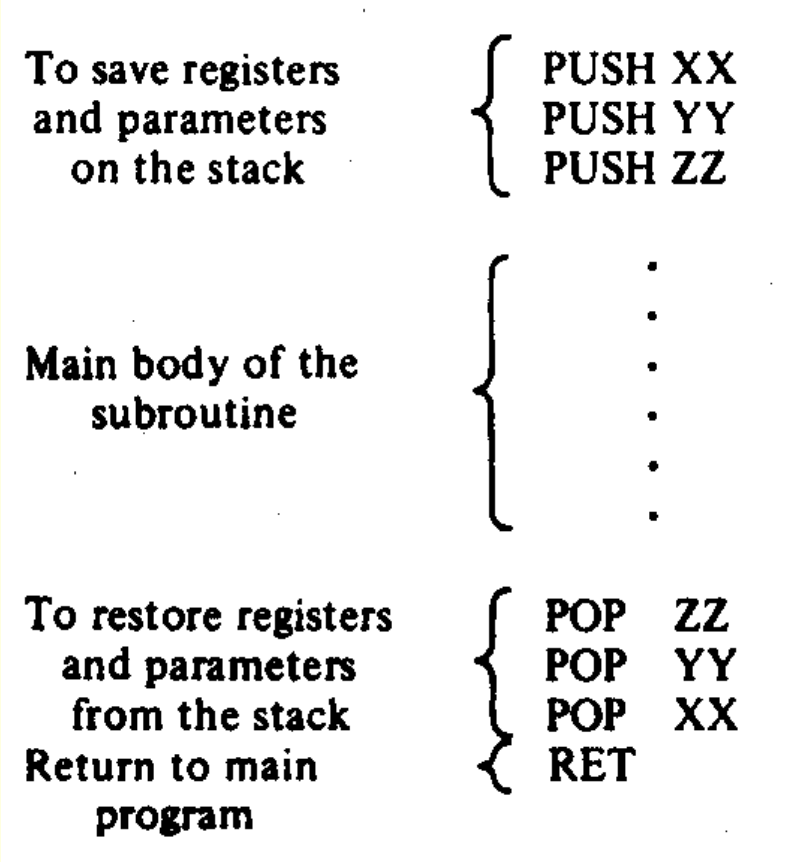
Algorithm:

Pop from stack: IP CS

if immediate operand is present: SP = SP + operand

Subroutines and Subroutine-Handling Instructions

- The PUSH and POP instructions



Subroutines and Subroutine-Handling Instructions

- *EXAMPLE:*
  - Write a procedure named SQUARE that squares the contents of BL and places the result in BX
- *Solution:*

```
;Subroutine:   SQUARE
;Description:  (BX) = square of (BL)

SQUARE  PROC   NEAR
        PUSH  AX          ;Save the register to be used
        MOV   AL,BL       ;Place the number in AL
        IMUL  BL          ;Multiply with itself
        MOV   BX,AX       ;Save the result
        POP   AX          ;Restore the register used
        RET
SQUARE  ENDP
```

String and String-Handling Instructions

- The basic 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) (DI) ← (DI) ± 1 or 2	None

String and String-Handling Instructions

- Move string – MOVSB, MOVSW
- Example – The block-move program using the move-string instruction

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

String and String-Handling Instructions

- Compare string and scan string – CMPSB/CMPSW, SCASB/SCASW
- Example – Block scan operation using the SCASB instruction

```
        MOV     AX,0
        MOV     DS,AX
        MOV     ES,AX
        MOV     AL,05
        MOV     DI,0A000H
        MOV     CX,0FH
        CLD
AGAIN:  SCASB
        LOOPNE  AGAIN
NEXT:
```

String and String-Handling Instructions

- Load and store string –LODSB/LODSW, STOSB/STOSW
- Example – Initializing a block of memory with a store string instruction

```
        MOV     AX,0
        MOV     DS,AX
        MOV     ES,AX
        MOV     AL,05
        MOV     DI,0A000H
        MOV     CX,0FH
        CLD
AGAIN:  STOSB
        LOOP    AGAIN
```

String and String-Handling Instructions

- REP string – REP (repeat prefixes)

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

String and String-Handling Instructions

- REP string – REP (repeat prefixes)
- Example – Initializing a block of memory by repeating the STOSB instruction

```
MOV     AX,0
MOV     DS,AX
MOV     ES,AX
MOV     AL,05
MOV     DI,0A000H
MOV     CX,0FH
CLD
REPSTOSB
```

String and String-Handling Instructions

- Autoindexing for string instruction – CLD and STD instructions

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

String and String-Handling Instructions

- *EXAMPLE:* Describe what happen as the following sequence of instruction is executed

```
CLD
MOV     AX, DATA_SEGMENT
MOV     DS, AX
MOV     AX, EXTRA_SEGMENT
MOV     ES, AX
MOV     CX, 20H
MOV      SI, OFFSET MASTER
MOV      DI, OFFSET COPY
REPMOVSB
```