CSC405 Microprocessor

8086 Microprocessor

Instruction Set (Part2)

String Instructions

- String in assembly language is a series of bytes stored sequentially in the memory location.
- String instructions operate on strings.
- By using these string instructions, the size of the program is considerably reduced.

String Instructions

- The **Source element** is taken from **Data Segment** **using** the **SI register**.
- The **Desitination element** is in **Extra Segment** **pointed** by the **DI register.**
- **SI &/or DI** are **incremented /decremented** after each operation depending upon the **direction flag "DF" in Flag Register**

8086 - Instruction Set - String Instructions

LODS: LODSB/LODSW (Load String)

- Load **AL** or **AX** Reg with a byte or word from **data segment**
- **Offset** of the **source** is in **SI**
- **SI** is **incremented /decremented** after each operation depending upon the **direction flag "DF" in Flag Register**
- Eg: LODSB ; AL ⟵ DS:[SI]

  ; SI ⟵ SI ± 1 : depending on DF

  LODSW ; AL ⟵ DS:[SI], AH ⟵ DS:[SI+1]

  ; SI ⟵ SI ± 2 : depending on **DF**

8086 - Instruction Set - String Instructions

STOS: STOSB/ STOSW (Store String)

- Store **AL** or **AX** into a byte or word in **Extra segment**
- **Offset** of the source in **Extra Segemnt** is in **DI**
- **DI** is **incremented /decremented** after each operation depending upon the **direction flag "DF" in Flag Register**
- Eg: STOSB, STOSW

8086 - Instruction Set - String Instructions

General mnemonic Op-code	Operand	Object code	Coding example: Mnemonic	Segment for memory access	Symbolic operation	Description
STOSB	None	AA	STOSB	Extra	ES:[DI] ← AL If DF = 0, DI ← DI + 1 If DF = 1, DI ← DI − 1	Transfer a byte or word from register AL or AX to the string element addressed by DI in the extra segment; if DF = 0, increment DI, else decrement DI; the flags are not affected
STOSW	None	AB	STOSW	Extra	ES:[DI] ← AL ES:[DI + 1] ← AH If DF = 0, DI ← DI + 2 If DF = 1, DI ← DI − 2	
STOS	*dest*[a]	AA	STOS MEMBES	Extra	Same as STOSB if DI = MEMBES	
		AB	STOS MEMWES	Extra	Same as STOSW if DI = MEMWES	
LODSB		AC	LODSB	Data	AL ← DS:[SI] If DF = 0, SI ← SI + 1 If DF = 1, SI ← SI − 1	Transfer a byte or word from the string element addressed by SI in the data segment to register AL or AX; if DF = 0, increment SI, else decrement SI; the flags are not affected
LODSW		AD	LODSW	Data	AL ← DS:[SI] AH ← DS:[SI + 1] If DF = 0, SI ← SI + 2 If DF = 1, SI ← SI − 2	
LODS	*source*[a]	AC	LODS MEMBDS	Data	Same as LODSB if SI = MEMBDS	
		AD	LODS MEMWDS	Data	Same as LODSW if SI = MEMWDS	

8086 - Instruction Set - String Instructions

MOVS: MOVSB/MOVSW (Move String)

- Used to transfer a word/byte from **data segment to extra segment**
- **Offset** : source is in SI and destination is in DI
- **SI &/or DI** are **incremented /decremented** after each operation depending upon the **direction flag "DF" in Flag Register**
- Eg: MOVSB ; *ES:[DI]* ⟵ *DS:[SI]*

  ; SI ⟵ SI ± 1 & DI ⟵ DI ± 1: depending on DF

  MOVSW *;{ES:[DI], ES:[DI+1]}* ⟵ *DS:[SI], DS:[SI+1]*

  ; SI ⟵ SI ± 2 & DI ⟵ DI ± 2: depending on DF

8086 - Instruction Set - String Instructions

General mnemonic Op-code	Operand	Object code	Coding example: Mnemonic	Segment for memory access	Symbolic operation	Description
MOVSB	None	A4	MOVSB	Data, extra	ES:[DI] ← DS:[SI] If DF = 0, DI ← DI + 1 SI ← SI + 1 If DF = 1, DI ← DI − 1 SI ← SI − 1	Transfer a byte or word from the string element addressed by SI in the data segment to the string element addressed by DI in the extra segment; if DF = 0, increment SI and DI, else decrement SI and DI; the flags are not affected
MOVSW	None	A5	MOVSW	Data, extra	ES:[DI] ← DS:[SI] ES:[DI + 1] ← DS:[SI + 1] If DF = 0, DI ← DI + 2 SI ← SI + 2 If DF = 1, DI ← DI − 2 SI ← SI − 2	
MOVS	*dest,source*[a]	A4	MOVS MEMBES, MEMBDS	Extra, data	Same as MOVSB if SI = MEMBDS and DI = MEMBES	
		A5	MOVS MEMWES, MEMWDS	Extra, data	Same as MOVSW if SI = MEMWDS and DI = MEMWES	

8086 - Instruction Set - String Instructions

SCAS: SCASB/ SCASW (Scan String byte or word)

- **Compare** the contents of **AL** or **AX** with a **byte or word in the Extra Segemnt**
- **Offset** of the byte or word in Extra Segemnt is in **DI**
- **DI** is **incremented /decremented** after each operation depending upon the **direction flag "DF" in Flag Register**
- Comparision done by **substracting** byte or word from **AL** or **AX**; result not stored; flag bits affected
- Eg: SCASB, SCASW

8086 - Instruction Set - String Instructions

CMPS: CMPSB/ CMPSW (Compare String)

- **Compare** a byte or **word in the Data Segment** with a byte or **word in the Extra Segemnt**
- **Offset** of the byte or word in Data Segment is in SI
- **Offset** of the byte or word in Extra Segemnt is in DI
- **SI &/or DI** is **incremented /decremented** after each operation depending upon the **direction flag "DF" in Flag Register**
- Comparision done by substraction ([DS] – [ES]), result not stored; flag bits affected
- Eg: CMPSB, CMPSW

kk

General mnemonic Op-code	Operand	Object code	Coding example: Mnemonic	Segment for memory access	Symbolic operation	Description
CMPSB		A6	CMPSB	Extra, data	DS:[SI] − ES:[SI]; update flags If DF = 0, DI ← DI + 1 SI ← SI + 1 If DF = 1, DI ← DI − 1 SI ← SI − 1	Subtract the byte or word of the destination string element addressed by DI in the extra segment from the byte or word of the source string element addressed by SI in the data segment; if DF = 0, increment DI and SI, else decrement SI and DI; the flags are updated to reflect the relationship of the destination operand to the source operand
CMPSW		A7	CMPSW	Extra, data	DS:[SI + 1:SI] − ES:[DI + 1:DI]; update flags If DF = 0, DI ← DI + 2 SI ← SI + 2 If DF = 1, DI ← DI − 2 SI ← SI − 2	
CMPS	*dest,source*[a]	A6	CMPS MEMBES, MEMBDS	Extra, data	Same as CMPSB if SI = MEMBDS and DI = MEMBES	
		A7	CMPS MEMWES, MEMWDS	Extra, data	Same as CMPSW if SI = MEMWDS and DI = MEMWES	
SCASB		**AE**	**SCASB**	**Extra**	AL − ES:[DI]; update flags If DF = 0, DI ← DI + 1 If DF = 1, DI ← DI − 1	Subtract the byte or word of the string element addressed by DI in the extra segment from AL or AX; if DF = 0, increment DI, else decrement DI; the flags are updated to reflect the relationship of the destination operand to the source operand
SCASW		**AF**	**SCASW**	**Extra**	AX − ES:[DI + 1:DI]; update flags If DF = 0, DI ← DI + 2 If DF = 1, DI ← DI − 2	
SCAS	*dest*[a]	**AE**	**SCAS MEMBES**	**Extra**	Same as SCASB if DI = MEMBES	

8086 - Instruction Set - String Instructions

REP(Repeat)

- Instruction – **prefix** used with string instructions
- Can be used only with string instructions
- Causes the instruction to be **repeated CX number of times**
- After every execution, SI and **DI reg are incremented /decremented** after each operation depending upon the **direction flag "DF" in Flag Register and CX is decremented**

General mnemonic		Object code	Coding example			
Op-code	Operand		Mnemonic	Segment for memory access	Symbolic operation	Description
REP		F3 AA	REP STOSB	Extra	STOSB; CX ← CX − 1 Repeat until CX = 0	The string instruction following the REP prefix is repeated until CX is decremented to 0
		F3 AB	REP STOSW	Extra	STOSW; CX ← CX − 1 Repeat until CX = 0	
		F3 A4	REP MOVSB	Extra, data	MOVSB; CX ← CX − 1 Repeat until CX = 0	
		F3 A5	REP MOVSW	Extra, data	MOVSW; CX ← CX − 1 Repeat until CX = 0	
REPE/REPZ[a]		F3 AE	REPZ SCASB	Extra	SCASB; CX ← CX − 1 Repeat if ZF = 1 and CX ≠ 0	Repeat the string operation if the scan or compare is equal (i.e., ZF = 1) *and* CX ≠ 0; decrementing CX does not affect the flags
		F3 AF	REPZ SCASW	Extra	As above except SCASW	
		F3 A6	REPZ CMPSB	Extra, data	As above except CMPSB	
		F3 A7	REPZ CMPSW	Extra, data	As above except CMPSW	
REPNE/REPNZ[a]		F2 AE	REPNE SCASB	Extra	SCASB: CX ← CX − 1 Repeat if ZF = 0 and CX ≠ 0	Repeat the string operation if the scan or compare is not equal (i.e., ZF = 0) *and* CX ≠ 0; decrementing CX does not affect the flags
		F2 AF	REPNE SCASW	Extra	As above except SCASW	
		F2 A6	REPNE CMPSB	Extra, data	As above except CMPSB	
		F2 A7	REPNE CMPSW	Extra, data	As above except CMPSW	

8086 - Instruction Set - String Instructions

REPZ/REPE(Repeat on ZERO/ Equal)

- **Conditional repeat** Instruction – prefix
- Behaves the same as REP instruction provided **ZF =1**
- It is used with CMPS instruction

REPNZ/REPNE(Repeat on No ZERO/ Equal)

- Conditional repeat Instruction – prefix
- Behaves the same as REP instruction provided **ZF = 0**
- It is used with SCAS instruction

8086 - Instruction Set – Transfer Control of Instructions

- Unconditional Jump Instructions
- Conditional Jump Instructions
- Loop Instructions
- Call and Return Instructions
- Software Interrupts

8086 - Instruction Set – Transfer Control of Instructions

I. **Near Branch**

- This is a Intra-Segment branch (within the segment)
- Needs only IP to be changed
- If the Near branch is in the range of **–128 to 128**, it is called as **short jump**

II. **Far Branch**

- This is a Inter-Segment branch (between the segments)
- ∴ both CS & IP needs to be changed

8086 - Instruction Set – Transfer Control of Instructions

```
ADDR    HEX CODES   LABELS     OP-CODE    OPERANDS      COMMENTS
0000    B3 04                  MOV        BL,04H        ;Load divisor to BL
0002    E5 06       REPEAT:    IN         AX,06H        ;Get data word
0004    F6 F3                  DIV        BL            ;Divide by four
0006    E7 9A                  OUT        9AH,AX        ;Output result
0008    E9 F7 FF               JMP        REPEAT        ;Repeat the cycle
000B
```

$$\begin{array}{rr} & 000BH \\ + & FFF7H \\ \hline 1 & 0002H \end{array}$$

8086 - Instruction Set – Transfer Control of Instructions

```
ADDR    HEX CODES   LABELS      OP-CODE     OPERANDS        COMMENTS
0000    B3 04                   MOV         BL,04H          ;Load divisor to BL
0002    E5 06       REPEAT:     IN          AX,06H          ;Get data word
0004    F6 F3                   DIV         BL              ;Divide by four
0006    E7 9A                   OUT         9AH,AX          ;Output result
0008    EB F8                   JMP SHORT REPEAT            ;Repeat the cycle
000A
```

```
MOV   BX,0002H      ;BX points at REPEAT
JMP   BX            ;Jump to 0002H
```

8086 - Instruction Set – Transfer Control of Instructions

General mnemonic		Object code	Coding example			
Op-code	Operand		Mnemonic	Segment for memory access	Symbolic operation	Description
JMP	*near target*	E9 — — [a]	JMP MEMN[b]	Code	IP ← MEMN	Transfer control to (near) target location within the segment; the addressing mode may be direct, memory indirect, or register indirect
		FF 26 00 10	JMP [MEMWDS][c]	Data	IP ← [MEMWDS + 1:MEMWDS]	
		FF 27	JMP [BX]	Data	IP ← [BX + 1:BX]	
		FF E0	JMP AX	Within CPU	IP ← AX	
JMP	SHORT *target*	EB — [d]	JMP SHORT MEMS[e]	Code	IP ← MEMS	Transfer control to the (short) target location; there are no indirect forms of this instruction
JMP	*far target*	EA 03 00 D3 9E	JMP FAR PTR MEMF[f]	Code	IP ← 0003H; CS ← 9ED3H	Transfer control to the far target location outside the segment
		FF 2E 05 10	JMP [MEMWWDS][g]	Data	IP ← [1006H:1005H]; CS ← [1008H:1007H]	
		FF 2F	JMP DWORD PTR[BX]	Data	IP ← [BX + 1:BX]: CS ← [BX + 3:BX + 2]	

8086 - Instruction Set – Transfer Control of Instructions

General mnemonic		Object code	Coding example			
Op-code	Operand		Mnemonic	Segment for memory access	Symbolic operation	Description
Jcond[h]	*short target*	73 — [d]	JNC MEMS	Code	If CF = 0, then IP ← MEMS	Transfer control to the short target address if the condition is true; all conditional jumps require short targets
JCXZ	*short target*	E3 — [d]	JCXZ MEMS	Code	If CX = 0, then IP ← MEMS	Transfer control to the short target address if CX = 0

JCXZ is a special type conditional jump that does not test the status flags.

It tests the contents of CX and transfers control to the target address if CX = 0

[a]These 2 bytes are the 2's complement offset between near mem location MEMN & the instruction immediately following the JMP instruction.

[b]MEMN is assumed to point to a near–mem location(i.e., within the segment).

[c]MEMWDS is assumed to point to the word beginning at location 1000H in the data segment. Brackets are optional as MEMWDS defines a word, not a location to jump to.

[d]This byte is the 2's complement offset between short mem location MEMS & the instruction immediately following the jump SHORT instruction.

[e]MEMS is assumed to point to short mem location. (i.e., +127 to -128 locations from the address of the instruction immediately following the JMP SHORT inst). The SHORT operator is optional if MEMS is already known by the assembler.

[f]MEMF is assumed to point to the far memory location 9ED3H:0003H

[g]MEMWWDS is assumed to point to the double word beginning at loc 1005H in the data segment. Brackets are optional as MEMWWDS defines a word, not a location to jump to.

[h]refer to table 2.15 for a list of testable conditions.

8086 - Instruction Set – Transfer C

- Conditional Jump Instructions
  - Performs short jump based on condition of status flag
  - Placed after Arithmetic and logical instructions

Mnemonic	Condition
	Signed Operations
JG/JNLE	Greater/not less nor equal ((SF ⊕ OF) + ZF) = 0
JGE/JNL	Greater or equal/not less (SF ⊕ OF) = 0
JL/JNGE	Less/not greater nor equal (SF ⊕ OF) = 1
JLE/JNG	Less or equal/not greater ((SF ⊕ OF) + ZF) = 1
JO	Overflow (OF = 1)
JS	Sign (SF = 1)
JNO	Not overflow (OF = 0)
JNS	Not sign (SF = 0)
	Unsigned Operations
JA/JNBE	Above/not below nor equal (CF ⊕ ZF) = 0
JAE/JNB	Above or equal/not below (CF = 0)
JB/JNAE	Below/not above nor equal (CF = 1)
JBE/JNA	Below or equal/not above (CF ⊕ ZF) = 1
	Either
JC	Carry (CF = 1)
JE/JZ	Equal/zero (ZF = 1)
JP/JPE	Parity/parity even (PF = 1)
JNC	Not carry (CF = 0)
JNE/JNZ	Not equal/not zero (ZF = 0)
JNP/JPO	Not parity/parity odd (PF = 0)

8086 - Instruction Set – Transfer Control of Instructions

- Conditional Jump Instructions – Example

```
MOV   BL,47H
IN    AL,36H
CMP   AL,BL
JE    MATCH
JA    BIG
JMP   SMALL
```

8086 - Instruction Set – Transfer Control of Instructions

- Loop Instructions

```
        LEA     SI,TABLE        ;Load SI with base address of TABLE
        MOV     CX,0100H        ;CX holds the number of bytes to output
AGAIN:  LODSB                   ;Data byte to AL and INC SI
        OUT     0A0H,AL         ;Output the data byte
        LOOP    AGAIN           ;Repeat until CX = 0
```

- LOOPE/LOOPZ – CX≠0 & ZF=1
- LOOPNE/LOOPNZ – CX≠0 & ZF=0
- LOOPE, LOOPNE – comparing 2 strings
- JCXZ – check if CX has not accidently become zero.

8086 - Instruction Set – Transfer Control of Instructions

General mnemonic			Coding example			
Op-code	Operand	Object code	Mnemonic	Segment for memory access	Symbolic operation	Description
LOOP	*short target*	E2 —[a]	LOOP MEMS [b]	Code	CX ← CX − 1 If CX ≠ 0, then IP ← MEMS	Decrement CX and transfer control to the short target address if CX ≠ 0
LOOPE[c] LOOPZ	*short target*	E1 —[a]	LOOPZ MEMS [b]	Code	CX ← CX − 1 If (CX ≠ 0)·(ZF = 1), then IP ← MEMS	Decrement CX and transfer control to the short target address if CX ≠ 0 AND the last instruction to affect the flags resulted in zero (ZF = 1)
LOOPNE[c] LOOPNZ	*short target*	E0 —[a]	LOOPNZ MEMS[b]	Code	CX ← CX − 1; If (CX ≠ 0)·(ZF = 0), then IP ← MEMS	Decrement CX and transfer control to the short target address if CX ≠ 0 AND the last instruction to affect the flags resulted in a nonzero result (ZF = 0)

8086 - Instruction Set – Transfer Control of Instructions

- [a]This byte is the 2's complement offset between short mem location MEMS & the instruction immediately following the LOOP instruction. The offset is limited to +127 to -128.
- [b]MEMS is assumed to point to short mem location. (i.e., +127 to -128 locations from the address of the instruction immediately following the LOOP inst.
- [c]either form of the mnemonic may be used.

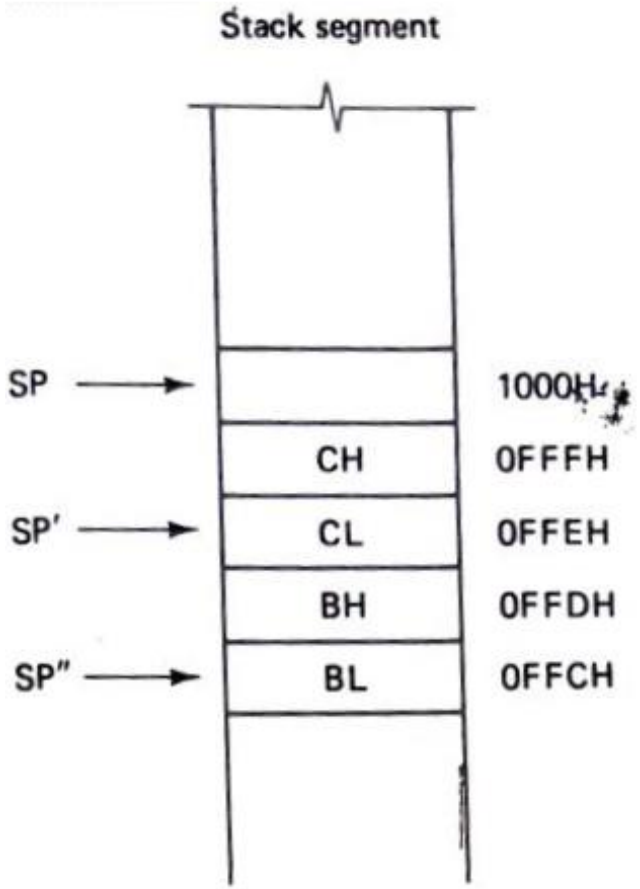
8086 - Instruction Set

- PUSH and POP

Addressing mode	Object code	Coding example: Mnemonic op-code operand	Segment for memory access	Symbolic operation	Description
PUSH *source*[a]	51	PUSH CX	Stack	SP ← SP − 2; [SP + 1] ← CH; [SP] ← CL	Decrement SP by 2 and transfer the word from the source operand to the top of the stack now pointed to by SP
	1E	PUSH DS	Stack	SP ← SP − 2; [SP + 1:SP] ← DS	
	FF 75 02	PUSH [DI + 2]	Stack, data	SP ← SP − 2; [SP + 1] ← [DI + 3]; [SP] ← [DI + 2]	
POP *dest*[a]	59	POP CX	Stack	CL ← [SP]; CH ← [SP + 1]; SP ← SP + 2	Transfer the word from the top of the stack pointed to by SP to the destination operand; increment SP by 2
	1F	POP DS	Stack	DS ← [SP + 1:SP]; SP ← SP + 2	
	8F 45 02	POP [DI + 2]	Data, stack	[DI + 3] ← [SP + 1]; [DI + 2] ← [SP]; SP ← SP + 2	
PUSHF none	9C	PUSHF	Stack	SP ← SP − 2; [SP + 1:SP] ← flags	Push the 16-bit flag word onto the stack
POPF none	9D	POPF	Stack	flags ← [SP + 1:SP]; SP ← SP + 2	Pop the top of the stack into the 16-bit flag word

8086 - Instruction Set

- Initialize SP to a high memory location as stack grows downwards
- Registers should be popped of the stack in the reverse order in which they are pushed in.
- PUSHF and POPF Instructions
  - useful with interrupts and subroutines as the entire state of machine can be saved and then restored later



PUSH CX
PUSH BX

8086 - Instruction Set – Transfer Control of Instructions

- **Call** and **Return** Instructions

```
DELAY    PROC    NEAR      ;Define near procedure
         PUSH    AX        ;Save AX on stack
         MOV     AX,0000   ;Set counter for 65536 cycles
REPEAT:  DEC     AX        ;Bump counter
         JNZ     REPEAT    ;Do 65536 times
         POP     AX        ;Restore AX
         RET               ;Return to main program
DELAY    ENDP
```

8086 - Instruction Set – Transfer Control of Instructions

General mnemonic		Object code	Coding example: Mnemonic	Segment for memory access	Symbolic operation	Description
Op-code	Operand					
Call	*near target*	E8 — —[a]	CALL MEMN[b]	Code	SP ← Sp − 2; [SP + 1:SP] ← IP; IP ← MEMN	IP is pushed onto the top of the stack and control is transferred within the segment to the near target address
		FF 16 00 10	CALL [MEMWDS][c]	Data	SP ← SP − 2; [SP + 1:SP] ← IP IP ← [1001H;1000H]	
		FF 15	CALL [DI]	Data	SP ← SP − 2; [SP + 1:SP] ← IP; IP ← [DI + 1:DI]	
		FF D7	CALL DI	Within CPU	SP ← SP − 2 [SP + 1:SP] ← IP; IP ← DI	
CALL	*far target*	9A 00 10 D3 09	CALL FAR PTR MEMF[d]	Code	SP ← SP − 2; [SP + 1:SP] ← CS; CS ← 09D3H; SP ← SP − 2; [SP + 1:SP] ← IP; IP ← 1000H	CS and IP are pushed onto the top of the stack and control is transferred to the new segment and far target address
		FF 1E 00 10	CALL [MEMWWDS][e]	Data	Same as above except: CS ← [1003H:1002H]; IP ← [1001H;1000H]	
		FF 1D	CALL DWORD PTR[DI]	Data	Same as above except: CS ← [DI + 3:DI + 2]; IP ← [DI + 1:DI]	

8086 - Instruction Set – Transfer Control of Instructions

General mnemonic		Object code	Coding example			
Op-code	Operand		Mnemonic	Segment for memory access	Symbolic operation	Description
RET	*n* (near)[f]	C3	RET	Stack	IP ← [SP+1:SP]; SP ← SP+2	The word at the top of the stack is popped into IP transferring control to this new address; RET is normally used to return control to the instruction following a near subroutine call; if included, the optional pop value (n) is added to SP
		C2 08 00	RET 8	Stack	IP ← [SP+1:SP]; SP ← SP+2+8	
RET	*n* (far)[f]	CB	RET	Stack	IP ← [SP+1:SP]; SP ← SP+2; CS ← [SP+1:SP]; SP ← SP+2	As above except that the double word at the top of the stack is popped into IP and CS transferring control to this new far address
		CA 08 00	RET 8	Stack	IP ← [SP+1:SP]; SP ← SP+2; CS ← [SP+1:SP]; SP ← SP+2+8	

8086 - Instruction Set – Transfer Control of Instructions

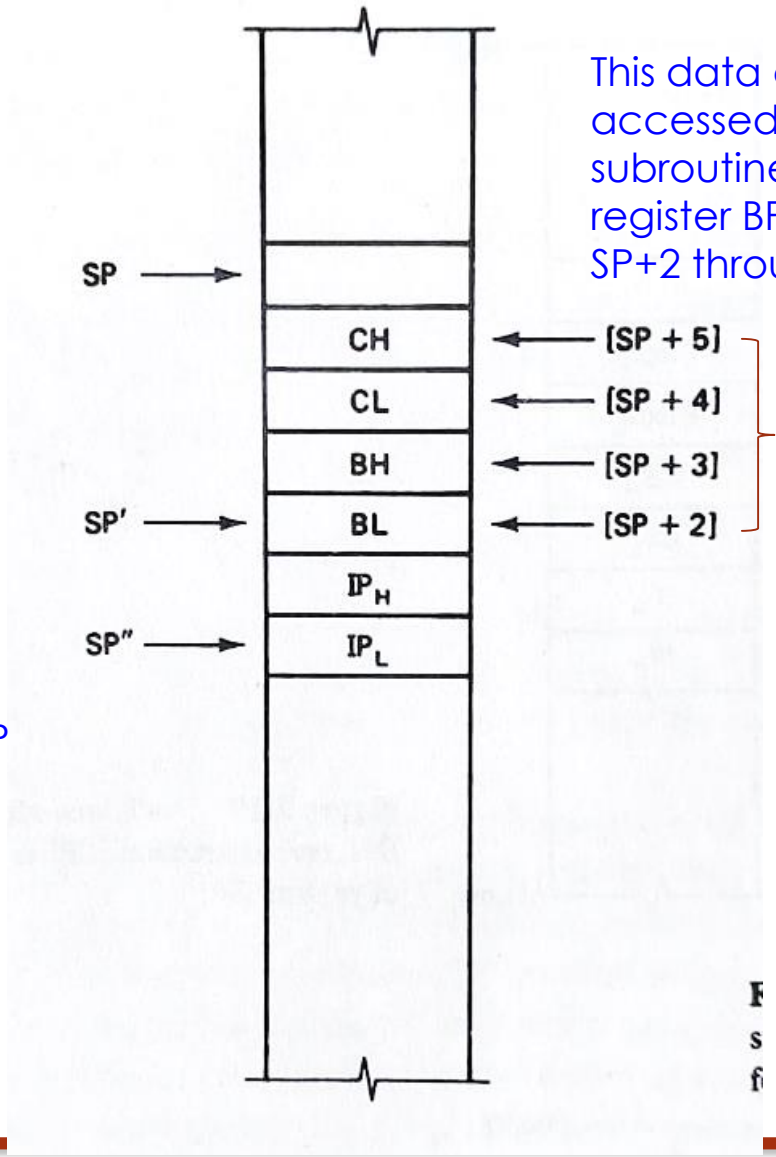
- [a]These 2 bytes are the 2's complement offset between near mem location MEMN & the instruction immediately following the CALL instruction.
- [b]MEMN is assumed to point to a near–mem location(i.e., within the segment).
- [c]MEMWDS is assumed to point to the word beginning at location 1000H in the data segment. Brackets are optional as MEMWDS defines a word, not a location to jump to.
- [d]FAR PTR indicates that MEMF is located in a different code segment. In this case its assumed that MEMF is points at 09D3H:1000H
- [e]MEMWWDS is assumed to point to the double word beginning at loc 10009H in the data segment. Brackets are optional as MEMWWDS defines a double word, not a location to jump to.
- [f]The same mnemonic is used for a near or far return. Assembler will generate the proper code based on the procedure definition statement.(near or far)

8086 - Instruction Set – Transfer Control of Instructions

- Call and Return Instructions

```
    PUSH CX
    PUSH BX
    CALL SUB
SUB: .
     .
     .
    RET 4
```

This POPs IP off the stack, incrementing SP to SP'.
The POP value of 4 then causes SP' to be incremented to SP(original value).



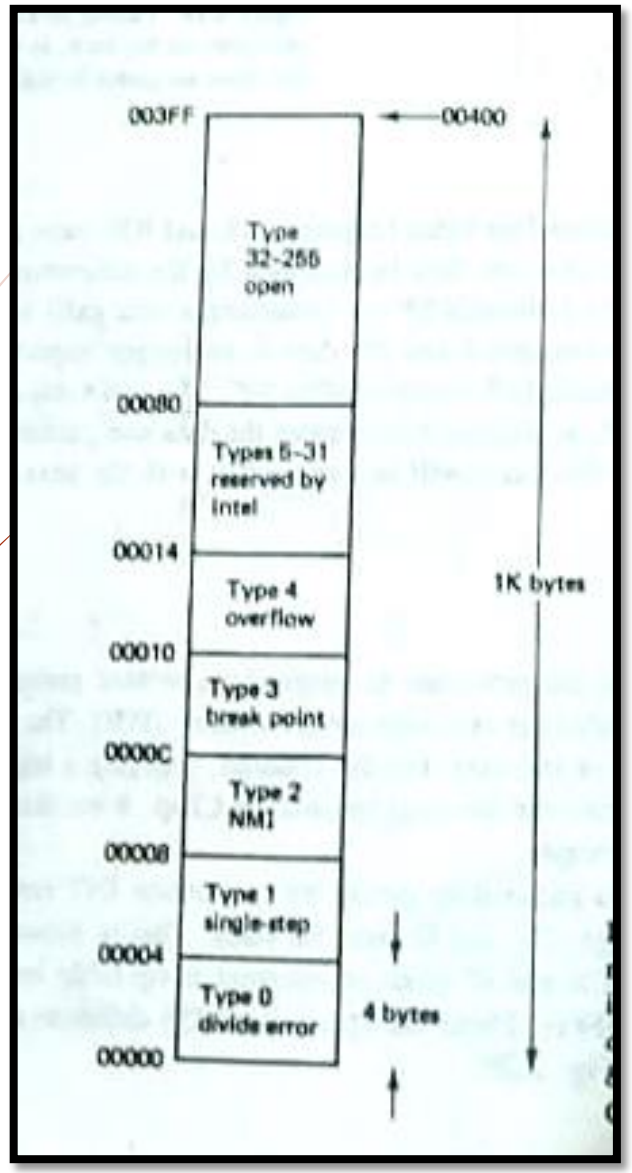
This data can be accessed by the subroutine using register BP to point to SP+2 through SP+5

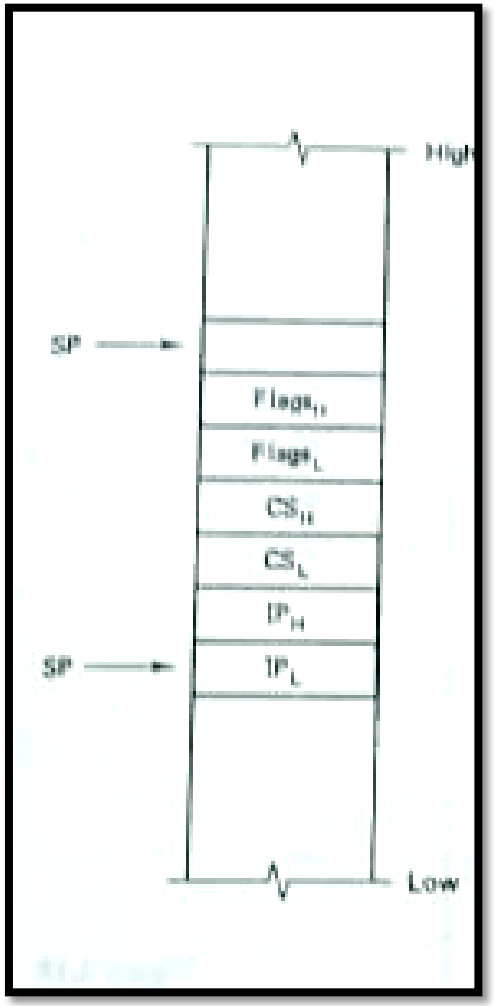
8086 - Instruction Set – Transfer Control of Instructions

- Software Interrupts are initiated by giving the instruction INT type
- In effect INT 23 performs a far call to the address stored in the double word at 0008CH Through 0008FH

General mnemonic: Op-code	General mnemonic: Operand	Object code	Mnemonic	Segment for memory access	Symbolic operation	Description
INT	*type*	CD 23	INT 23H	Stack and interrupt jump table at 00000–003FFH	SP ← SP − 2; [SP + 1:SP] ← flags IF ← 0; TF ← 0; SP ← SP − 2; [SP + 1:SP] ← CS; CS ← [0008FH:0008EH];[a] SP ← SP − 2 [SP + 1:SP] ← IP; IP ← [0008DH:0008CH][b]	Save the flag, CS, and IP registers on the stack and transfer control to the far address stored in the double word beginning at absolute address *type* * 4
INTO	none	CE	INTO	Stack and interrupt jump table at 00000–003FFH	If OF = 1, then SP ← SP − 2 [SP + 1:SP] ← flags; IF ← 0; TF ← 0; SP ← SP − 2; [SP + 1:SP] ← CS; CS ← [00013H:00012H];[a] SP ← SP − 2; [SP + 1:SP] ← IP; IP ← [00011H;00010H][b]	If an overflow condition exists (OF = 1), a type 4 interrupt is executed
IRET	none	CF	IRET	Stack	IP ← [SP + 1:SP]; SP ← SP + 2; CS ← [SP + 1:SP]; SP ← SP + 2; flags ← [SP + 1:SP]; SP ← SP + 2	Transfer control back to the point of interrupt by popping the IP, CS, and flag registers from the stack; IRET is normally used to exit any interrupt procedure whether activated by hardware or software

Software Interrupts





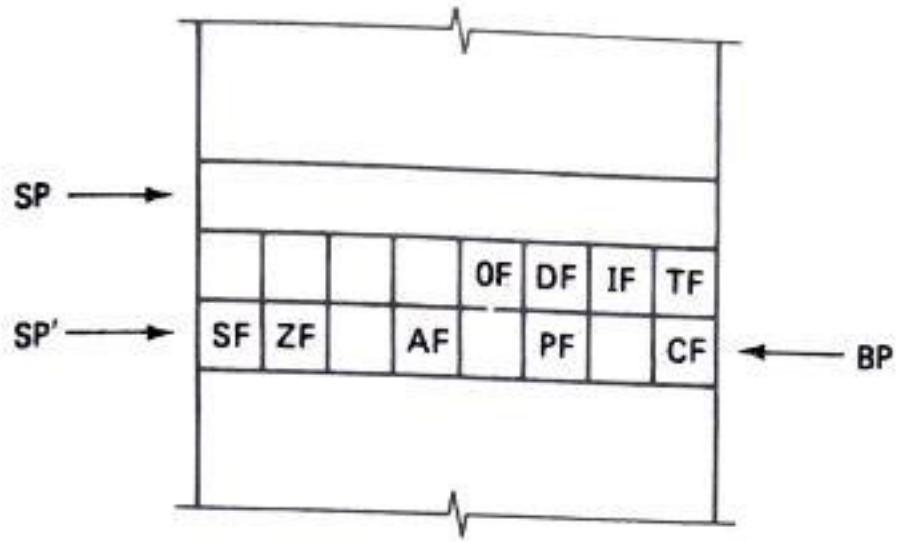
8086 - Instruction Set – Processor Control Instructions

- Used to control the operation of the processor and set or clear the status indicators.
- CF, DF and IF **can be** set/cleared. CF **can be** inverted.
- PF, AF, ZF and OF **cannot** be accessed with specific control instructions.
- DF, IF and TF are processor control bits rather than status indicators.

8086 - Instruction Set – Processor Control Instructions

- Used to control the operation of the processor and set or clear the status indicators.
- CF, DF and IF can be set/cleared. CF can be inverted.
- PF, AF, ZF and OF can not be accessed with specific control instructions.
- DF, IF and TF are processor control bits rather than status indicators.
- Setting the Trap Flag

```
PUSHF
MOV BP,SP
OR BYTE PTR[BP+1],01H      ;Set bit 0– TF
POPF
```



8086 - Instruction Set – Processor Control Instructions

General mnemonic		Object code	Coding example			
Op-code	Operand		Mnemonic	Segment for memory access	Symbolic operation	Description
STC	None	F9	STC	Within CPU	CF ← 1	Set carry flag
CLC	None	F8	CLC	Within CPU	CF ← 0	Clear carry flag
CMC	None	F5	CMC	Within CPU	CF ← $\overline{CF}$	Complement carry flag
STD	None	FD	STD	Within CPU	DF ← 1	Set direction flag (auto decrement for string instructions)
CLD	None	FC	CLD	Within CPU	DF ← 0	Clear direction flag (auto increment for string instructions)
STI	None	FB	STI	Within CPU	IF ← 1	Set interrupt flag (enabling interrupts on the INTR line)
CLI	None	FA	CLI	Within CPU	IF ← 0	Clear interrupt flag (disabling interrupts on the INTR line)

8086 - Instruction Set – Processor Control Instructions

- **HLT**
  - This instruction stops the processor and causes it to enter idle loop.
  - Once halted, processor can be restarted only via hardware interrupt or a system reset.
- **NOP** – no operation.
  - Used in debugging
  - Used to pad time delay routines

8086 - Instruction Set – Processor Control Instructions

General mnemonic		Object code	Coding example			
Op-code	Operand		Mnemonic	Segment for memory access	Symbolic operation	Description
HLT	None	F4	HLT	Within CPU	None	Halt
WAIT	None	9B	WAIT	Within CPU	None	Enter wait state if $\overline{TEST}$ line = 1
LOCK	*instruction*	F0 A1 00 10	LOCK MOV AX,MEMWDS[a]	Data	None	Output line $\overline{LOCK}$ = 0 while the instruction following LOCK executes; used to prevent coprocessors from accessing the bus during a particular instruction
NOP	None	90	NOP	Within CPU	None	No operation
ESC	*number, source*	DE 0E 00 10	ESC 31H,MEMWDS[a]	Data	Data bus ← [MEMWDS]	Place the contents of the memory source operand on the data bus and execute a NOP; the first operand identifies a particular escape instruction to be executed by a coprocessor

8086 - Instruction Set – Processor Control Instructions – 3 instructions that go with special coprocessors.

- **WAIT**
  - Used to synchronize 8086 with 8087 NDP via $\overline{\text{TEST}}$ input
- **LOCK**
  - This instruction is executed when it is important that no other processor attempt to control the system buses.
  - LOCK is in effect for the single instruction only.
- **ESC**
  - Used as a prefix to the coprocessor instructions.
  - The coprocessor is activated by the ESC prefix and reads the operand from data bus and begin execution