Contents :

Intel Barry B. Brey-Ch 1

2102024 - sharafat

Chapter - 1

1-6

1) Charles Babbage
2) Joseph Jacquard, a Frenchman who used it as input to a weaving machine.
3) Hollerith
4) Monroe
5) Military applications
6) ENIAC
7) intel 8008
8) Countess of Lovelace worked with Charles Babbage in the development of software for his analytic engine.
9) Grace Hopper
10) Von Neuman machine is a type of system where instructions are also stored in memory.
11) 8008
12) 100 million

13) 8086

14) 16M byte

15) 4G

16) 1993

17) 1995

18) late 2000

19) Pentium 4

20) MIPS = Millions of Instructions per second

21) CISC = Complex instruction set computer

22) 1 or 0

23) 10^3 bytes

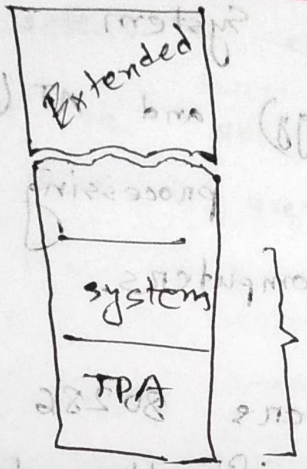
24) 2^{10} "

25) 2^{20} "

26) 2^{30} "

27) Depends on the per page size

28) TPA and system area



29) 2G

30) 640K

31) 2G

32) 1M

33) 1T

34) 80386.

35) extended memory

36) The system BIOS is a collection of programs stored in either a read-only (ROM) or flash memory that operates many of the I/O devices connected to our computer system.

37) DOS = Disk Operating System

38) XT (Extended technology) and AT (Advanced Technology) are two processing power measurement of computers..

8086 used XT where 80286 introduced AT. AT was significantly faster, could handle twice as much data as XT.

39) VESA local bus or VL bus interfaces disk and video to the microprocessor.

40) 16

41) Universal Serial Bus (USB) is a serial data path with twisted pair of cables which is intended to connect peripheral devices such as keyboard, mouse, etc.

42) AGP or Advanced Graphics port is a type of bus that transfers data between the video card and microprocessor.

43) XMS refers to extended memory system.

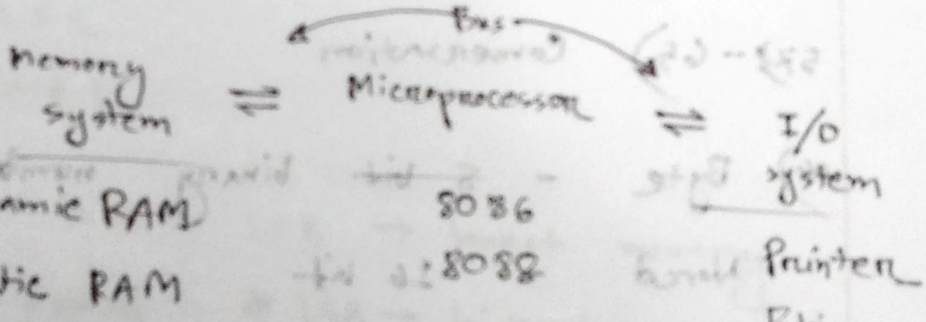
44) SATA or serial ATA interface is a bus that is used for hard disk drives and the PCI Express bus for video card.

45) TPA

46) 64K

47) [same as 36]

48)



49) Doing the computations...

50) Address, data, control

51) Address

52) $\overline{MRDC}$

53) IO Read control

54) Memory read

55) DB - Define Byte
DQ - Define Quadword
DW - Define word
DD - Define Doubleword

56) char - 8 bit character
short - 16 bit integer
int - 32 bit integer
float - 32 bit floating point
double - 64 bit floating point

57 - 65) Conversation

66) Byte - 8 bit binary number
Word - 16 bit
Doubleword - 32 bit

67) conversation

68) The ASCI for enter key is carriage return or CR. It generally inserts a new line to the document.

69) Unicode: Unicode is a system used by windows based applications to store alphanumeric data.

70) LINE DB 'what time it is?'

71-72) Conversation

73) DATA DB -34

75) (a) 3412 High to low memory

(b) 22A1

(c) 00B1

76) Little endian = LSB → lowest memory

Big endian = LSB → highest memory

77) DATA DW 123AH

78) (a) 102

packed - 0102H

Unpacked - 01 00 02 H

79-80) Convention

Chapter - 02

27 1) Registers that are visible to programmers and can be accessed by programming, is known as program visible registers.

2) 16

3) 80386

4) EBX

5) CX

6) Instruction pointer (IP/EIP) holds the offset of the next instruction to be run.

7) Increment / Decrement

8)
```
  FFH
  01H
------
 100H
```

~~Yes overflow~~ No overflow will occur if it is 8 bit addition. In result we will get 1 as carry flag

Here FFH will be considered as -1.

9) odd

10) I (Interrupt) flag

11) ~~3086~~ 8038- core2

12) It holds the base address

13) (a) 10000H - 1FFFFH

(b) 12340H - 223FFH

14) csX10H + IP

15) 1M

16) DI

17) EAX, EBX, ECX, EDX, ESI, EDI

18) SS + SP/ ESP

19) SS

20) segment X10H + offset

21) (a) DS = 2000H

EAX = 0000 3000H

20000
3000
23000H

22) all area

23) any location

24) The segment register works as a se
to select a descriptor from the desc
table

25) 2^{13} / 8192

26) starting → 00000 H
ending → 01000 H

27) starting → 01000000 H
ending → 01000000 + 0FFFF H

28) starting → 00280000H
ending → 00280000H + 00010 FFF H

29)

0000	0000	0010	0000
selector = 4		TI	RPL

30-31) same

32) 64k

33)

80286

0000H	FEH	21
00000H	001FH	

Access rights

P	DPL	S	E	ED/C	R/W	A
1	11	1	1	1	1	0

& FEH

34)

03	G 1	D 1	0	AV 1	0	F2	00
0000						8FFF	

Access right

P	DPL	S	E	ED/C	R/W	A
1	11	1	0	0	1	0

35) GDTR (Global Descriptor Table locator)

36) Through a descriptor at Global table

37) The microprocessor access the descriptor table using selector. Its base address, limit, and access rights are loaded into the program invisible segment register cache.

38) Registers that programmers can't access.

39) GDTR contains the base address of global descriptor table

40) 4K

41) CR0

42) 1024

43) 4K times

44)

49) Flat mode: Only segment + no offset

50) 1T

Chapter 03

1) (a) MOV AX, BX

It moves the content of BX into AX

(d) moves 32 bit content

(e) moves 64 bit content

2) AH, AL, BH, BL, CH, CL, DH, DL

3) AX, BX, CX, DX, SP, BP, SI, DI

4) EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI

5) RAX, RBX, RCX, RDX, RSP, RBP, RSI, RDI

6) CS, DS, ES, SS, FS, GS

7) Types doesn't match

8) Segment to segment not allowed directly

9) (a) MOV EDX, EBX

(b) MOV CL, BL

(c) MOV BX, SI

(d) MOV AX, DS

(e) MOV AH, AL

(f) MOV R10, R8

10) (a) MOV AL, 12H

(b) MOV AX, 123AH

....

11) Older assembler uses # Modern assembler uses like B/H suffix

12) Selects single segment model

13) .CODE directive

14) symbolic memory address

15) opcode

16) letter/some special characters

17) exit to DOS

18) Tiny model creates .COM program while small model creates .EXE program

19) .STARTUP is used to load data segment registers with the segment address of DS

20) DS X 10H + 2000H

21) memory address

22) (a) DS X 10H + 1234H

(b) " + 0300H

(c) " + 0400H

23) Memory-to-memory transfer not allowed.

24) MOV BYTE PTR [2000H], 16

25-27) same

28) MOV BX, DATA - copies the word

MOV BX, OFFSET DATA - copies offset

29) (a) (2000H X 10H) + 1000H + 0100H

(b) DS

(c) SS

30) MOV AL, [BX][SI]

is equivalent to

MOV AL, [BX+SI]

So it will work

31) (a) (12000 + 010C)H

(b) 12000 + 250 + 100 = 12350H

32) (a) DS X 10H + LIST + BX

(b) DS X 10H + LIST + BX + SI

33) (a) SS X 10H + BP + 200H

(b) SS X 10H + BP + SI − 200H

34) BP / SP

35) same

36)
```
FIELDS   STRUC
         F1   WORD   ?
         F2   WORD   ?
         F3   WORD   ?
         F4   WORD   ?
         F5   WORD   ?
FIELDS   ENDS
```

37) MOV AX, FIELDS.F3

38) 1- Direct
2- Relative
3- Indirect

39) 5 bytes

opcode + offset (low) + offset (high) + segment (low) + segment (high)

40) Intersegment → new segment (new → cs + IP)

Intrasegment → same segment (only IP)

41) signed 16 bit can reach ± 32k which means it can access the 64k

42) 32

43) far jump changes both cs and IP

44) short jump (1B)

45) (a) short

(b) Near

(c) short (diff 2h)

(d) Far

46) JMP BX

47) JMP WORD PTR [TABLE]

48) 2B

49) Pushes the content to the stack

50) AX, CX, DX, BX, SP, BP, SI, DI

51) Just like 80, but for 32

52) PUSHFD - Flag

53) No

Chapter 4

1) Opcode

2) D = Direction
   W = Word

3) addressing mode, whether the operator is register / memory / memory address

4) DL

5) Depends on W, if W = 1 it works for 32

6) DS:[BX + DI]

7) (a) SP - SS
   (b) EBX - DS
   (c) DI - DS
   (d) EBP - SS
   (e) SI - DS

8) MOV AL, BX

9) MOV BX, [SI + 4C00H]

10) 8B 77 02

11) Same

12) REX is used for 64 bit

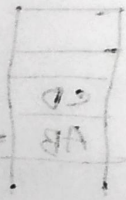
13) Moving data into CS isn't allowed

14)
```
mov AX, 1000H
mov DS, AX
```

15) 32

16) PUSH RAX

17) CS

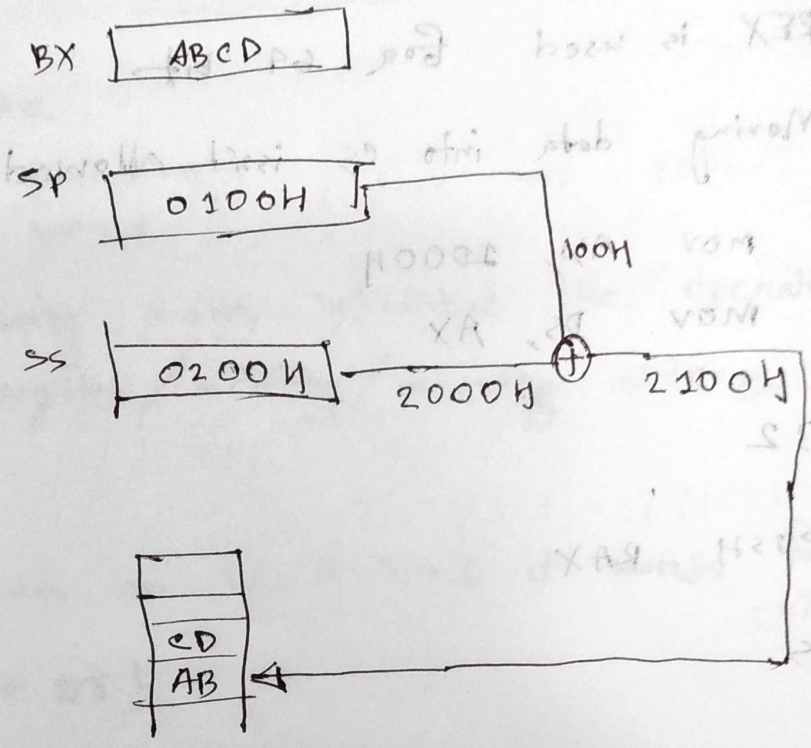


18) AX, CX, DX, BX, SP, BP, SI, DI

19) 32 bit (like 18)

20)
- (a) AX to stack
- (b) stack to ESI
- (c) [BX] to stack
- (d) Flags to stack
- (e) to stack to DS
- (f) 4 to stack

21)



22) same

23) 2

24) Any possible combination

25) mov → contents
LEA → offset

26) same

27) mov with offset

28) segment → DS
offset → BX

29) segment location

30) so easy, just mov operation

31) Direction flag: For string operation wh
the DI and SI increases/decreases

32) STD → set direction
CLD → clear direction

33) movs, cmps

34) LODSB loads DS:SI into AL

35) uses RAX

36) [DS:SI] to output (DX port)

37) SOSW = AX → [ES:DI]

38) MOV CX, 12
MOV LEA SI, SOURCE
LEA DI, DESI
CLD
REP MOVSB

39) Iterates until cx is 0

40) XCHG EBX, ESI

41) DX

42) LAHF = Load AH with flag
SAHF = Store " " "

43)
```
.MODE SMALL
.STACK 100h
.DATA
    Table DB 30h, ..... 39h
    Res DB 10 DUP(?)
.CODE
MAIN PROC
    mov ax, @data
    mov ds, ax

    mov bx, offset Table
    mov di, offset Res

    mov cx, 10
    mov al, 0

ConvertLoop:
    XLATB
    mov [di], al  | inc
    inc di
    loop convertloop
```

44) AL ← DS : [BX + AL]

45) Input port to AL

46) Output port (outputs one bit)

47) Using non-default segment

48) MOV AH, ES:[BX]

49) XCHG

50) statement for assembler

51) conditional move

52) (a) DB - Byte

(b) DW - word

(c) DD - Doubleword

53) LIST 1 DB 30 DUP(?)

54) Constant

55) enable 686 instruction

56) memory model

57) DS

58) Depends on segment name

59) terminates and return code to AL

60) PROC main

main ENDP

62) Store Proc

```
mov [DI], AL
mov [DI+1], AL
mov [DI+2], AL
mov [DI+3], AL
```

Store ENDP

Chapter 5

1) (a) ADD AX, BX

(b) ADD AL, 12H

(c) ADD EBP, EDI

(d) ADD CX, 22H

(e) ADD AL, [SI]

(f) ADD FROG, CX

(g) ADD RCX, 234H

2) Different size

3) Segment register are not allowing for arithmetic result store directly

4) c - carry - 0

a - auxiliary - 0

s - sign - 0

z - zero - 0

o - overflow - 0

5) MOV DH, AL
ADD DH, BL
ADD DH, CL
ADD DH, DL
ADD DH, AH

6) same

7-8) same

9) ADC DX, BX

10) INC SP

11) Size not defined. Correct one will be

INC BYTE PTR [BX]

12) (a) SUB CX, BX

(b) SUB DH, 0EEH

...

13) C = 0
A = 1
Z = 0
O = 0
S = 1

14) MOV BX, AX
SUB BX, DI
SUB BX, SI
SUB BX, BP

15) DEC EBX

16) Subtract with borrow

17) SUB - stores result
CMP - only changes flag

18) AX

19) DX : AX

20) If upper half exists,
then C = 1, O = 1

21) RDX : RAX

22) MOV DL, 5
MOV AL, DL
MUL DL
MOV BL, AL
MUL BL

14) MOV BX, AX
SUB BX, DI
SUB BX, SI
SUB BX, BP

15) DEC EBX

16) Subtract with borrow

17) SUB - stores result
CMP - only changes flag

18) AX

19) DX : AX

20) If upper half exists,
then C = 1, D = 1

21) RDX : RAX

22) MOV DL, 5
MOV AL, DL
MUL DL
MOV BL, AL
MUL BL

23) MUL – Unsigned

IMUL – signed

24) BX = DX × 100 H

25) AX

26) AX

27) RAX

28) ① division by zero

② Overflow

29) DIV = unsigned

IDIV = signed

30) AH

31) RAX

32)
```
MOV AH, 0
MOV AL, BL
DIV CL
ADD AL, AL
MOV DL, AL
MOV DH, 0
```

33) DAA
DAS
AAA
AAS
AAM
AAD

34) AAM converts into unpacked BCD digits

35) AAA, AAS, AAM, AAD

36)
```
MOV CX, 5
MOV SI, 10000
Next_digit:
    XOR DX, DX
    DIV SI
    ADD AL, 30H
    MOV [BX], AL
    INC BX
    MOV AX, DX
    MOV DX, 0
    MOV SI, SI/10
    LOOP Next_digit
```

37) (B)
```
ADD DX, BX
DAA
ADD CX, AX
DAA
```

38) No

39) (a) AND BX, DX

40) (b) AND DH, 0EAH

(c) AND DI, BP

(d) AND EAX, 1122H

(e) AND [BP], CX

(f) AND DX, [SI-8]

(g) AND WHAT, AL

40)

MOV BH, DH

AND BH, 1FH

41) (a) OR AH, BL

(b) OR ECX, 88H

(c) OR SI, DX

(d) OR BP, 112H

(e) OR [RBX], RCX

(f) OR AL, [BP+40]

(g) OR WHEN, AH

42) MOV SI, DI
OR SI, 001FH

43) (a) XOR AH, BH
(b) XOR CL, 99H
(c) XOR DX, DI
(d) XOR RSP, 1A23H
(e) XOR [EBX], DX
(f) XOR DI, [BP + 60]
(g) XOR DI, WELL

44) (a) BTS AX, 0
BTS AX, 1
BTS AX, 2
BTS AX, 3
(b) AND AX, 1FFFH
(c) XOR AX, 0380h

45) AND stores result
but TEST doesn't

46) TEST CH, 4

47) NOT → 1's complement

NEG → 2's complement

48) (a) SHR DI, 3

(b) SHL AL, 1

(c) ROL AL, 3

(d) RCR EDX, 1

(e) SAR DH, 2

49) Accumulator vs memory

50) ES

51) D=0 → increment

D=1 → decrement

52) Repeat while equal

53) If not equal (ZF=1, CX=0)

54) Compares string

[DS:SI] with [ES:DI]

55)
```
MOV   CX, 300H
MOV   AL, 66H
LEA   DI, LIST
CLD
REPNE SCASB
```

56) 43H is displayed

Chapter- 06

1) Short jump transfers control within -128 to 127 bytes from the current position

2) near

3) far

4) far

5) For 32-bit mode
its around 2G

6) (a) Near
(b) Short
(c) Far

7) symbolic name of an address

8) IP

9) CS + IP

10) near

11) JUMP DI – register indirect

JMP [DI] – memory indirect

12) JMP [DI] – only offset (near jump)

JMP FAR PTR [DI] – far jump

13) C, A, O, S, Z

14) CF = 0 and ZF = 0

15) ~~OF = 0~~ overflow flag, OF = 1

16) JG, JGE, JLE, JL, JO, JNO, JNS, JE/JZ, J

17) JA, JAE, JB, JBE, JC, JNC, JE/JZ, JNE/J

18) JA, JBE

19) If CX = 0

20) SETZ AL

21) CX

22) ECX

23) RCX

24)
```
MOV AX, ES
MOV DI, OFFSET DATA2
MOV AL, 00H
MOV CX, 150H
CLD
STOSB
LOOP $-2
```

25) LOOPE only loops if the prev. operation generates equal flag

26)
```
CMP AL, 3
JNE NEXT
ADD AL, 2
NEXT:
```

27)
```
MOV CX, 100H
MOV SI, OFFSET BLOCK_START
MOV AL, 42H
MOV DI, 0 ; UP count
MOV BX, 0 ; DOWN count

COMPARE_LOOP:
    CMP [SI], AL
    JA IS_UP
    JB IS_DOWN
    JMP NEXT_BYTE
```

```
IS-UP:
    INC DI
    JMP NEXT_BYTE

IS_DOWN:
    INC BX

NEXT_BYTE:
    INC SI
    LOOP COMPARE_LOOP

MOV UP, DI
MOV DOWN, BX
```

28)

```
MOV SI, OFFSET BLOCKA
MOV DI, OFFSET BLOCKB

.REPEAT
    LODSB
    STOSB

.UNTIL AL == 00H
```

29) infinity loop

30)

```
MOV SI, OFFSET BLOCKA
MOV DI, OFFSET BLOCKB
CLD
```

```
.WHILE   AL != 12H
        LODSB
        ADD  AL, [DI]
        MOV  [DI], AL
        INC  DI
.ENDW
```

31) Breaks a loop

32) Procedure : Procedure is a reusable section of the code. They generally performs one task. It is stored once in memory but can be used as much as necessary.

33) Near call : Transfers flow within the code segment. changes only IP

Far call ; Transfers to a new block. changes cs + IP.

34) RET (Return)

35) RET removes the return address from the stack so the program returns.

36) By using NEAR or FAR

37) PROC

38)
```
CUBE PROC NEAR
    MOV AX, CX
    MUL CX
    MUL CX
    MOV CX, AX
    RET
CUBE ENDP
```

39) RET 6 is to clean up parameters that were pushed onto the stack by the caller before the procedure was called.

40)
```
CALC PROC NEAR USES SI DI DX CX
    MOV AX, DI
    MUL SI
    MOV CX, 100H
    DIV CX
    RET
CALC ENDP
```

41)

```
SUMS  PROC  NEAR  USES  EBX  ECX  EDX
      MOV   EDI, 0
      ADD   EAX, EBX
      ADC   EAX, ECX
      ADC   EAX, EDX

      SETNC AL
      MOV   EDI, 0
      RET
SUMS  ENDP
```

42) Interrupt is a hardware-initialized function call or a hardware-initiated procedure

43) INT, INTO, INT 3

44) 256

45) Interrupt = 4 byte pointer
- ↳ offset
- ↳ segment

46) 0, refers to the divide errors

47) RET: RET only restores the instruction pointer

IRET: Used as an exception handler.

~~Also~~ Also pops code segment and flag register

48) 32-bit return

49) 64-bit return

50) INTO only interrupts when overflow

51) Address: 40H x 4 = 100H

Se location:

100H – 103H

52) STI → Set Interrupt flag

CLI → Clear interrupt flag

53) Wait instruction

54) Bound interrupt will interrupt a program if the value in the register or memory location under the test is above or below the bounderies.

55) 16

56) BP → stack