Syllabus (/syllabus/4/microprocessor/)

Old Ques (/old-question/4/microprocessor/)

Internal Architecture and Features of 8086

Addressing Modes of 8086

Assembly Language Programming

Segment and Offset Address

EXE and COM Programs

Assembling, Linking and Compiling

Example Programs of 8086

Introduction to Microprocessor (/note/microprocessor/introduction-to-microprocessor/)

Programming with 8085 Microprocessor (/note/microprocessor/programming-with-8085-microprocessor/)

Microprocessor System (/note/microprocessor/microprocessor-system/)

Interrupt Operations (/note/microprocessor/interrupt-operations/)

Advanced Topices (/note/microprocessor/advanced-topices/)

Programming with 8086 Microprocessor

Internal Architecture and Features of 8086

Features of 8086 microprocessor

- It is a 16-bit microprocessor.
- It can address 1 MB of memory.
- It can pre-fetch upto 6 instruction bytes from memory and queues them.
- The data bus is of 16-bit and address bus is of 20-bits.
- It is divided into two separate units i.e. BIU and EU.

Internal Architecture of 8086 Microprocessor

us Interface Unit (BIU):

- It sends out addresses, fetch instructions from memory, read data from memory or orts and write data to memory or ports.
- It handles all transfers of data and address on the buses from EU.
- It has two parts: instruction queue and segment registers.
- BIU is capable to store up to 6 bytes of instructions with FIFO manner in a queue. he EU simply reads instruction from queue. So, the overlapping of instruction etch with execution, called pipelining is possible.
- It contains address used to produce 20 bit address. Four segment registers are sed to hold upper 16 bits of starting address of four memory segments at a articular time.
- Code Segment (CS) contains the base or start of current code segment.
- Instruction Pointer (IP) contains the offset from this address to next instruction yte to be fetched.
- Data Segment (DS) contains address of program's data segment which is used to locate data.
- Stack Segment (SS) contains address of program's stack segment.
- Extra Segment (ES) is used by string to handle memory addressing.

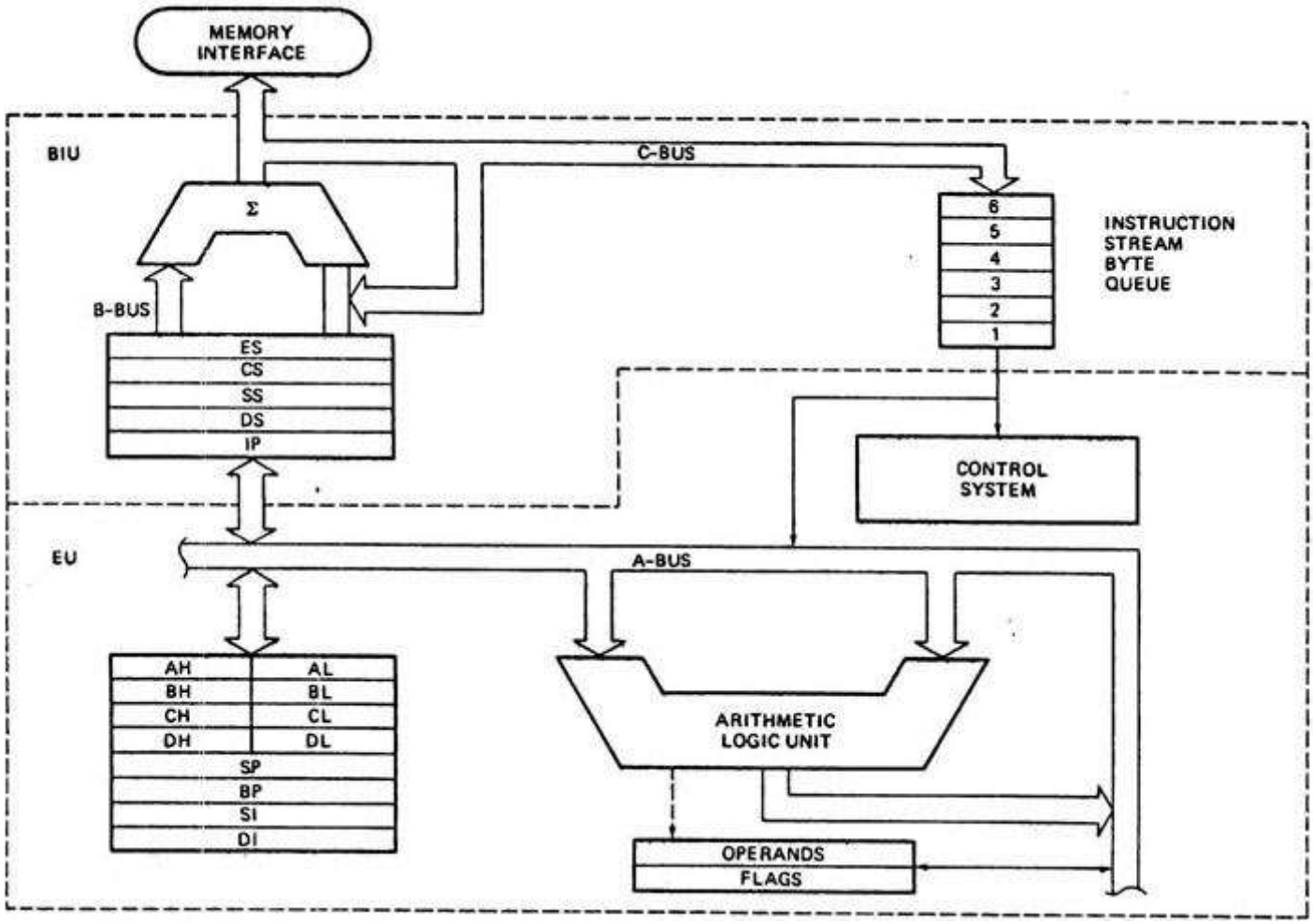
xecution Unit (EU):

- It decodes and executes the instructions.
- It contains ALU, control unit and a number of registers.
- It has nine 16-bits registers (AX, BX, CX, DX, SP, BP, SI, DI and flag).
- SP and BP are used to access data in stack segment; contains offset addresses.

- SP is used as an offset from current stack segment.
- BP contains offset address to current stack segment.
- SI and DI are used for indexed addressing.
- There are nine 1-bit flags, six of which are status flag and three are control flags.

-|-|-|-|O|D|I|T|S|Z|-|AC|-|P|-|CY

- If arithmetic overflow occurs, O flag is set.
- D flag is used by string manipulation instruction. If D is 0, string is processed from lowest address to highest address and if D is 1, string is processed from highest address to lowest address.
- If I is set, maskable interrupts are recognized otherwise ignored.
- If T is set, a trap is generated after execution of each instruction.



Addressing Modes of 8086

Immediate Addressing:

- Data is the part of instruction itself.
- Eg: MOV CX, 4929H

ADD AX, 2387H

Direct Addressing:
- Effective address of memory location is written in the instruction.
- Eg: MOV AX, [1592H]

Register Addressing:
- Register contains data in an instruction.
- Eg: MOV CX, AX

Register Indirect Addressing:
- The effective address of memory location is stored in a register through an offset.
- Eg: MOV AX, [BX]

Index Addressing:
- The operands offset address is found by adding contents of SI or DI register and 8 or 16 bits displacements.
- Eg: MOV BX, [SI + 16 bit]

Base Addressing:
- The operand offset address is given by sum of contents of base register and 8bit/16bit displacement.
- Eg: MOV DX, [BX + 04]

Base Index Addressing:
- The operand offset address is computed by sum of base register and index register.
- Eg: MOV AX, [AX +DI]

Base Index with Displacement Addressing:
- The operand offset address is computed by sum of base register, index register and 8 bit or 16 bit displacement.
- Eg: MOV AX, [BX + DI + 08]

Assembly Language Programming

8086 Instruction Set

1. Arithmetic Instructions

ADD reg(8bit)/mem(8bit), reg(8)/mem(8)/immediate
ADD reg(16bit)/mem(16bit), reg(16)/mem(16)/immediate
ADC reg/mem, reg/mem/immediate
SUB reg(8bit)/mem(8bit), reg(8)/mem(8)/immediate
SUB reg(16bit)/mem(16bit), reg(16)/mem(16)/immediate
SBB reg/mem, reg/mem/immediate
MUL reg/mem (Unsigned multiplication)
IMUL reg/mem (Signed multiplication)
DIV reg/mem (Unsigned division)
INC/DCC reg/mem (increment or decrement)
NEG (2's complement)
AAA
AAS
AAM
AAD
DAA
DAS

2. Logical Instructions

AND, OR, XOR
ROL, ROR
RCL, RCR
SHL, SHR
SAR, SAL
CMP
TEST

3. Data Transfer Instructions

MOV
LDS, LEA, LES, LSS
XCHG
IN, OUT

4. Flag Operations

CLC, CLD, CLI, STC,STD, STI
CMC
LAHF, SAHF, PUSHF, POPF

5. Stack Operations

PUSH reg(16)
POP reg(16)

6. Loop Instructions

LOOP, LOOPE, LOOPZ, LOOPNE, LOOPNZ

7. Branching Instructions

CALL, RET, INT, IRET, JMP, RETN, RETF
JA, JAE, JB, JBE, JC, JNC, JE, JNE, JZ, JNZ, JG, JNG, JL, JNL, JO, JS, JNS, JP, JPE, JPO, JNP

8. Type Conversion

CBW, CWD

9. String Instruction

MOVS/ MOVSB/ MOVSW
CMPS/ CMPSB/ CMPW
LODS/ LODSB/ LODW
REP

Operators

- Operator can be used in operand using immediate address/data.

Arithmetic: +, -, *, /
Logical: AND, OR, XOR, NOT
SHL and SHR
[ ] //Index operator
HIGH //returns higher byte of an expression
LOW
OFFSET
SEG

PTR
Segment Override
LENGTH
SIZE

Segment and Offset Address

- Segment in a program is a special area containing code, data and stack.
- A segment register is of 16 bits in size and contains starting address of a segment.
- The distance in bytes from the segment address to another location within the segment is called offset.

Eg: Let SA is 038E and OA is 0032H,
SA : OA
038E : 0032
038E * 10 + 0032
03912H (Physical address)

EXE and COM Programs

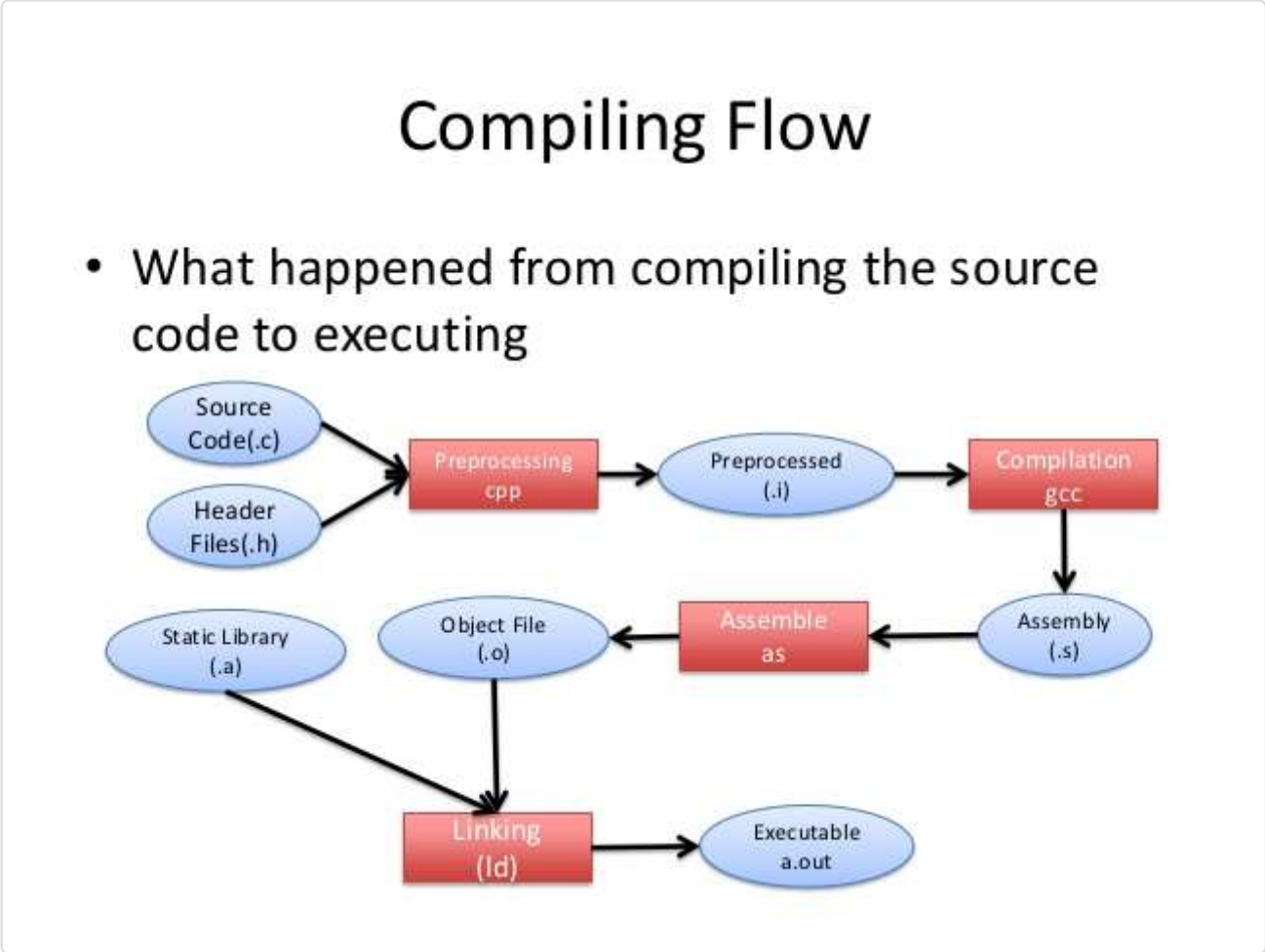
EXE and COM Programs

- Both of them are executable files.
- EXE deals with dynamic reallocation of memory while COM file has limited memory size allocation.
- EXE file contains a header but COM file does not have a header.
- EXE file can contain more than one segment but COM file has to contain only a single segment.
- The size of COM file is limited to 64K.

Assembling, Linking and Compiling

- The assembler translates a program written in assembly language into an object program. This process is called assembling.
- After assembling of source file, it forms object file (.obj) and list file (.lst).

- The list file



produces listing of source and object code and error diagnostics.
- The linker links all the object files and library files together to form an executable file and map file.
- The map file (.map) contains relative location and size of each segment.
- The DOS loader is used to execute an executable program.

Example Programs of 8086

1. Write ALP to read a text from keyboard, convert into uppercase and display on cleared screen.

```
.model small
.stack 100

.data
String label byte
Maxlen db 20
```

```
Actlen db ?
Str db 20 dup(?)

.code
MAIN PROC FAR
MOV AX, @DATA
MOV DS, AX
LEA DX, STRING
MOV AH, 0AH
INT 21H
CALL CLR_SCR
MOV CL, ACTLEN
MOV CH, 00H
MOV BX, 00H

STEP: MOV DL, STR[BX]
CMP DL, ‘a’
JB N1
CMP DL, ‘z’
JA N1
SUB DL, 20H
MOV AH, 02H
N1: INT 21H
INC BX
LOOP STEP
MOV AX, 4C00H
INT 21H
MAIN ENDP

CLR_SCR PROC NEAR
MOV AH, 0600H
MOV BH, 07H
MOV CX, 0000H
MOV DX, 184FH
```

```
INT 10H
RET
CLR_SCR ENDP

END MAIN
```

---

2. Write ALP to read string and display only alphabetic characters on cleared screen.

```
.MODEL SMALL
.STACK 100

.DATA
STRING LABEL BYTE
MAXLEN DB 60
ACTLEN DB ?
STR DB 60 DUP(?)

.CODE
MAIN PROC FAR
MOV AX, @DATA
MOV DS, AX
LEA DX, STRING
MOV AH, 0AH
INT 21H
CALL CLR_SCR
MOV CL, ACTLEN
MOV CH, 00H
MOV BX, 00H

STEP: MOV DL, STR[BX]
CMP DL, ‘A’
JB SKIP
CMP DL, ‘Z’
JA N1
JB SHOW
```

```
N1: CMP DL, ‘a’
JB SKIP
CMP DL, ‘z’
JA SKIP

SHOW: MOV AH, 02H
INT 21H

SKIP: INC BX
LOOP STEP
MOV AH, 4C00H
INT 21H
MAIN ENDP

CLR_SCR PROC NEAR
MOV AH, 0600H
MOV BH, 07H
MOV CX, 0000H
MOV DX, 184FH
INT 10H
RET
CLR_SCR ENDP

END MAIN
```

---

3. Write ALP to read string, display each word in new line in cleared screen, count no of words and display the count.

```
.MODEL SMALL
.STACK 100

.DATA
STRING LABEL BYTE
MAXLEN DB 60
ACTLEN DB ?
STR DB 60 DUP(?)
```

```
.CODE
MAIN PROC FAR
MOV AX, @DATA
MOV DS, AX
MOV ES, AX
LEA DX, STRING
MOV AH, 0AH
INT 21H
CALL CLR_SCR
MOV CL, ACTLEN
MOV CH, 00H
MOV BX, 00H
MOV AL, 00H

NEXT: MOV DL, STR[BX]
CMP DL, ‘ ’
JE NEW
MOV AH, 02H
INT 21H
JMP SKIP

NEW: INC EL
CALL NEW_L

SKIP: INC BX
LOOP NEXT
MOV DL, EL
MOV AH, 02H
INT 21H
MOV AX, 4C00H
INT 21H
MAIN ENDP

CLR_SCR PROC NEAR
MOV AH, 0600H
MOV BH, 07H
MOV CX, 0000H
```

```
MOV DX, 184FH
INT 10H
RET
CLR_SCR ENDP

NEW_L PROC NEAR
MOV AH, 0601H
MOV BH, 07H
MOV CX, 0000H
MOV DX, 184FH
INT 10H
RET
NEW_L ENDP

END MAIN
```

---

4. Write ALP to read string and count number of vowels, consonants, numerals and other characters and display the count.

```
.MODEL SMALL
.STACK 100

.DATA
STRING LABEL BYTE
MAXLEN DB 60
ACTLEN DB ?
STR DB 60 DUP(?)
VOWEL DB 0
CONSONANT DB 0
NUMERAL DB 0
OTHER DB 0

.CODE
MAIN PROC FAR
MOV AX, @DATA
MOV DS, AX
MOV ES, AX
```

```
LEA DX, STRING
MOV AH, 0AH
INT 21H
MOV EL, 00H
MOV CL, ACTLEN
MOV CH, 00H
MOV BX, 00H

NEXT: MOV DL, STR[BX]
CMP DL, '0'
JB N1
CMP DL, '9'
JA N1
MOV EL, NUMERAL
INC EL
MOV NUMERAL, EL
JMP N2

N1: CMP DL, 'A'
JB OTH
CMP DL, 'Z'
JB ALP
CMP DL, 'a'
JB OTH
CMP DL, 'z'
JA OTH

ALP: CMP DL, 'A'
JE VOW
CMP DL, 'a'
JE VOW
CMP DL, 'E'
JE VOW
CMP DL, 'e'
JE VOW
CMP DL, 'I'
```

```
JE VOW
CMP DL, ‘i’
JE VOW
CMP DL, ‘O’
JE VOW
CMP DL, ‘o’
JE VOW
CMP DL, ‘U’
JE VOW
CMP DL, ‘u’
JE VOW
MOV EL, CONSONANT
INC EL
MOV CONSONANT, EL
JMP N2

VOW: MOV EL, VOWEL
INC EL
MOV VOWEL, EL
JMP N2

OTH: MOV EL, OTHER
INC EL
MOV OTHER, EL

N2: LOOP NEXT

MOV DL, VOWEL
MOV AH, 02H
INT 21H
MOV DL, CONSONANT
MOV AH, 02H
INT 21H
MOV DL, NUMERALS
MOV AH, 02H
INT 21H
```

```
MOV DL, OTHER
MOV AH, 02H
INT 21H

MOV AX, 4C00H
INT 21H
MAIN ENDP

END MAIN
```

---

5. Write ALP to sort array of ten numbers stored in memory. Display after sorting.

```
.MODEL SMALL
.STACK 64

.DATA
ARR 11H, 20H, 9H, 5H, 16H, 22H, 12H, 14H, 8H, 17H

.CODE
MAIN PROC FAR
MOV AX, @DATA
MOV DS, AX
LEA DI, ARR
MOV CX, 09H

NEXT: CALL PERFORM
LOOP NEXT

LEA DI, ARR
MOV DL, [ARR]
MOV AH, 02H
INT 21H

MOV AX, 4C00H
INT 21H
MAIN ENDP
```

```
PERFORM PROC NEAR
PUSH CX
COM: MOV CX, 00H
MOV AL, [DI + CX]
MOV BL, [DI + CX + 1]
CMP AL, BL
JNC SKIP
PUSH AL
PUSH BL
POP AL
POP BL
MOV [DI + CX], AL
MOV [DI + CX + 1], BL

SKIP: INC CX
CMP CX, 0AH
JNE COM
POP CX
RET
PERFORM ENDP

END MAIN
```

Twitter

Tweets by @egnitenotes

Egnite Notes
@egnitenotes
Complete notes of OOP with C++egnitenotes.com/note/object-or…

Sep 13, 2017

Egnite Notes

Privacy Policy (/privacy-policy/)

Cookie Policy (/cookie-policy/)

Copyright Policy (/copyright-policy/)

User Agreement (/user-agreement/)

About Us (/about-us/)

Contact Us (/contact-us/)

Follow Us

(https://www.facebook.com/eginitenotes/)

(https://twitter.com/egnitenotes)