Lecture # 13

EQU vs TEXTEQU

- EQU is more general in that it allows numeric constants as well as text constants. EQU also explicitly states that a text value can be changed after declaration.

- TEXTEQU, on the other hand, only deals with text literals.
  - double quoted text,
  - literals proceeded by % , and
  - the values of macros.

```
name EQU expression
name EQU symbol
name EQU <text>
```

Unlike the = directive, a symbol defined with EQU cannot be redefined in the same source code file.

```
matrix1   EQU    10 * 10
matrix2   EQU   <10 * 10>
.data
M1 WORD matrix1
M2 WORD matrix2
```

The assembler produces different data definitions for **M1** and **M2**. The integer expression in **matrix1** is evaluated and assigned to **M1**. On the other hand, the text in **matrix2** is copied directly into the data definition for **M2**:

```
M1 WORD   100
M2 WORD   10 * 10
```

```
rowSize = 5
count   TEXTEQU  %(rowSize * 2)
move    TEXTEQU  <mov>
setupAL TEXTEQU  <move al,count>
```

Therefore, the statement

```
setupAL
```

would be assembled as

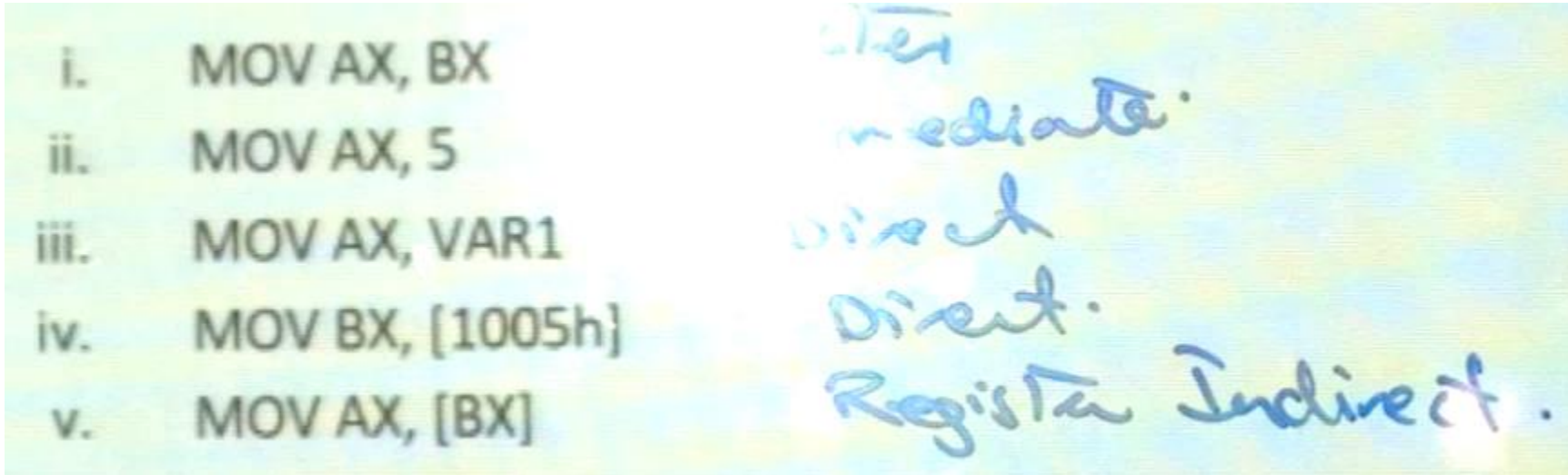
```
mov al,10
```

A symbol defined by TEXTEQU can be redefined at any time.

Midterm # 1 - Solution

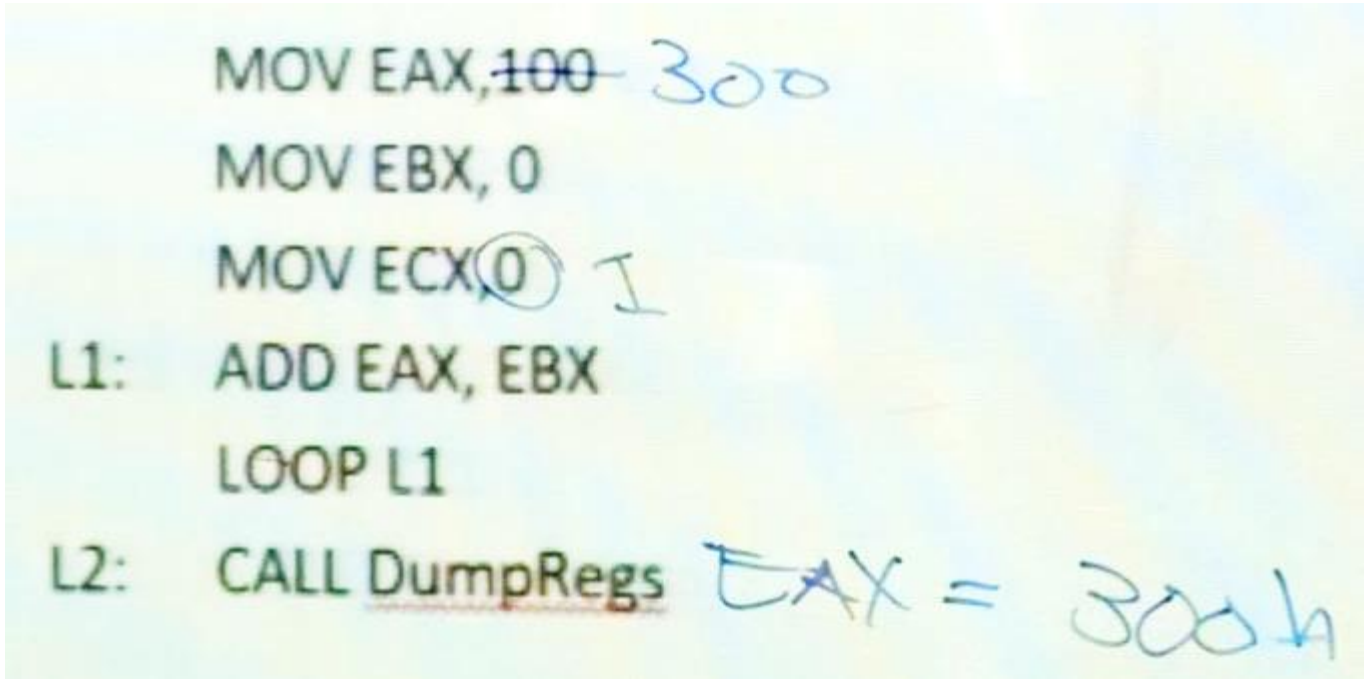
Q1 Answer the following questions:

a) Identify the addressing modes (type of operands) of the following instructions:



Q1 Answer the following questions:

b) Modify the following code snippet such that EAX contains 300 at label L2.



Q1 Answer the following questions:

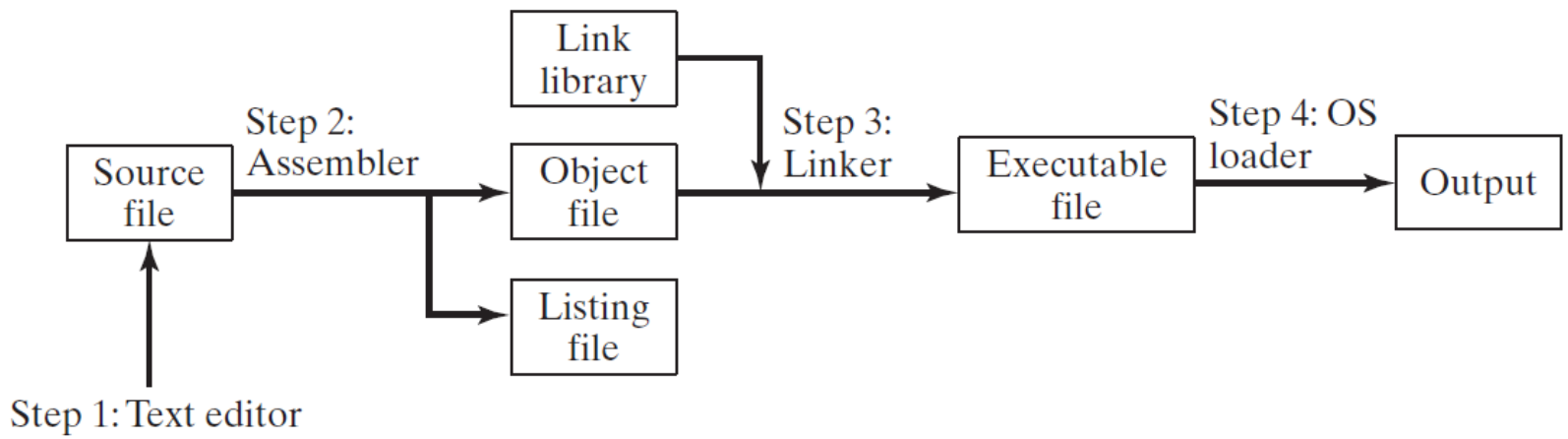
c) How the number CEF826F8h is stored in 1) big-endian order and 2) little-endian order.

The low byte is F8 and high byte is CE.

In little endian the low byte occupies low memory address and high byte the high address.
In big endian, the low byte occupies high memory address and high byte the low address.

d) Draw the "assemble-link-execute cycle" and number each step to show sequence of operations.

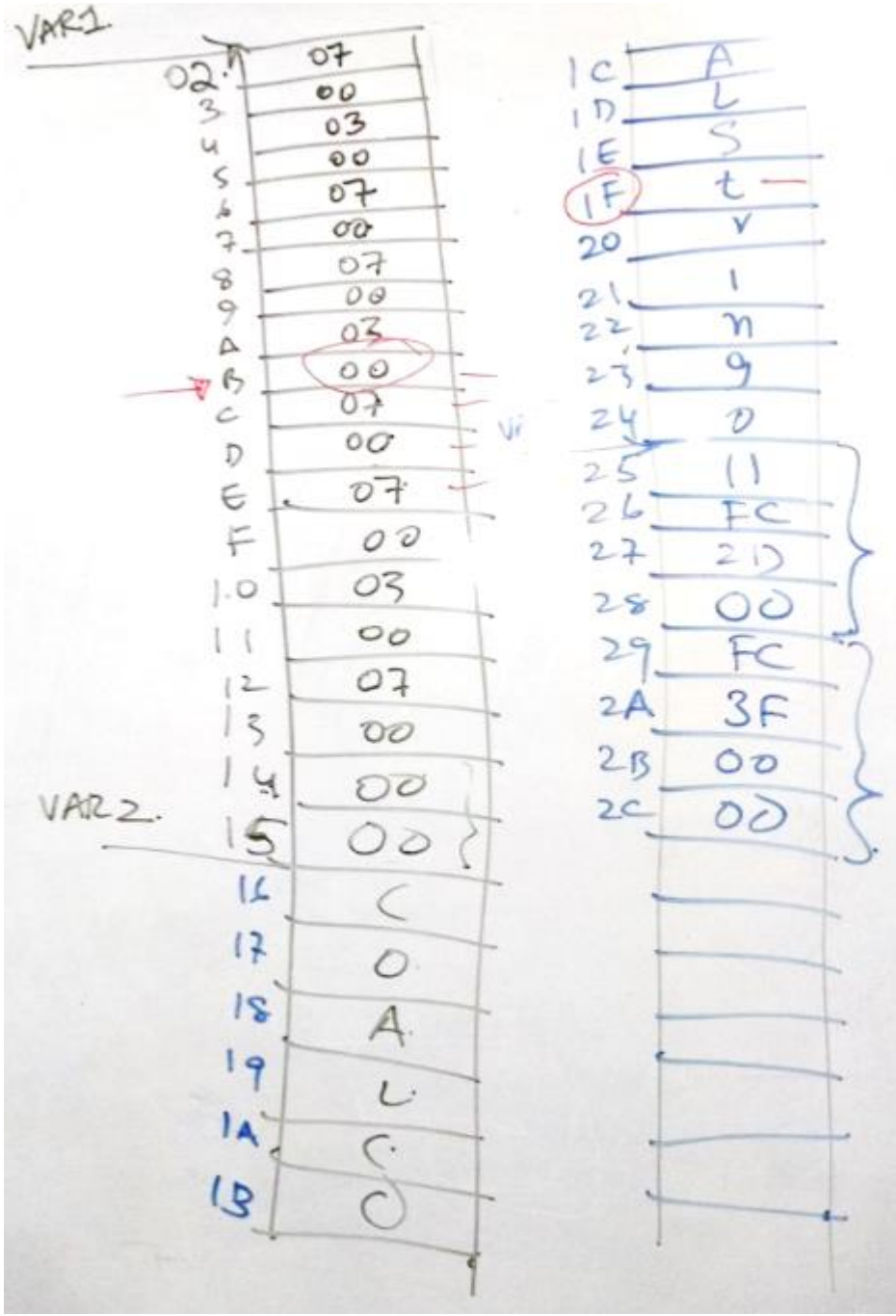
FIGURE 3–1 Assemble-Link-Execute Cycle.



Q2 Given the code snippet below, assuming the given data segment starts at **00000002h**, answer the following questions.

```
.data
      var1 WORD 3 DUP (7,3, 7), 0
      var2 BYTE 2 DUP ('COAL'), 'String',0
      var3 DWORD 2DFC11h, 3FFCh
```

a) Draw neat memory layout for the given data section showing byte by byte addresses and contents.



b) What does the following instruction result in?

```
MOV AX, WORD PTR var3
```

AX = FC11

c) What does **ESI** contain after the following instruction is executed?

```
MOV ESI, OFFSET var2+9
```

ESI = 0000001Fh

d) Write code that would swap second initializer of **var3** with second initializer of **var1**.

```
MOV AX, [var3+4]                ; AX = 3FFC
MOV WORD PTR [var1+2], AX
```

e) What does **EAX** contain when the following code is executed?

```
MOV ESI, OFFSET var1+9
MOV EAX, [ESI]
```

EAX = 07000700h

Q3 Identify and explain the type of error (if any) in the given instruction.

```
.data
 ARRAY1 DWORD     50 DUP (19)
 VAR1  BYTE  10
```

i. `MOV [BX], [SI]` → Both memory operated are not allowed

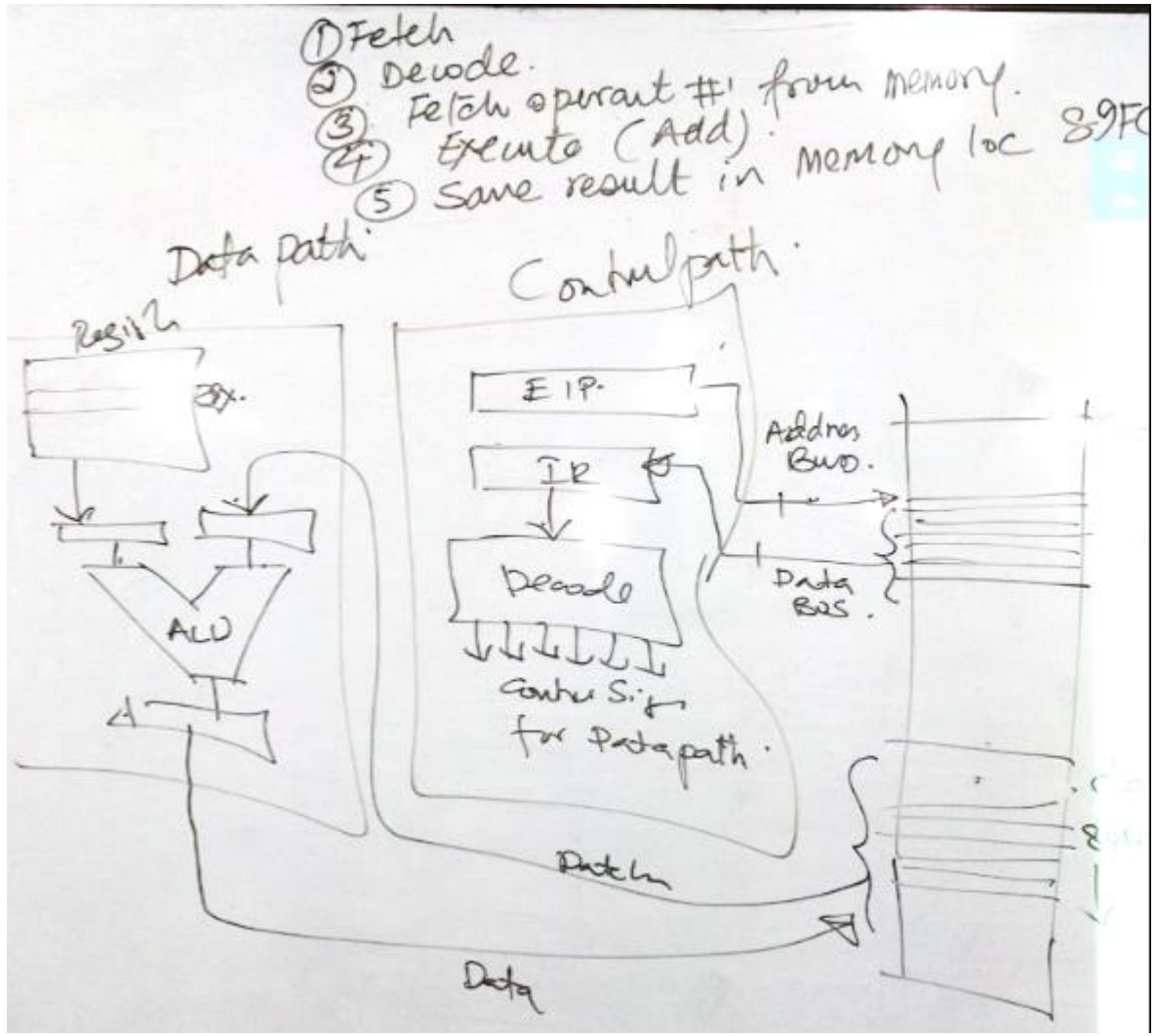
ii. `MOV EIP, 40` → EIP can not be changed by MOV instruction. Only jmp, call, ret change it.

iii. `MOV AX, [ARRAY1 + 1]` → No errors. A word is read into AX.

iv. `INC [ESI]` → Use PTR directive to identify the size of operation which will be incremented

v. `MOV EAX, WORD PTR VAR1` → Size of both operands must be save

Q4 Draw a simplified CPU block diagram, and show steps describing execution of x86 instructions **ADD [89FCh], EBX**. Show related memory locations with addresses, registers, and other element inside the CPU.



Q5 Write assembly code snippet for the following C code, that produces same results.

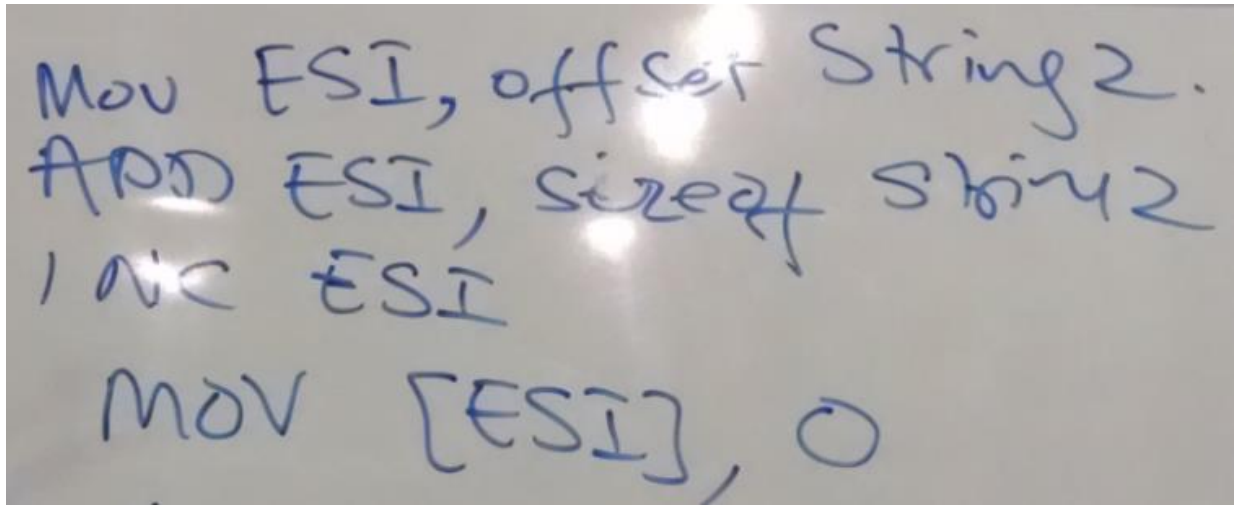
```
int a, b=20, c=30, f;
for (a=100, a != 0; a--)
     b = c + a;
     f = b + 2;
```

```
    MOV ECX, 100          ; a is mapped to ECX
    MOV EDX, C            ; c will not be changed
L1: MOV EBX, 0
    ADD EBX, ECX
    MOV A, ECX            ; update a
    ADD EBX, EDX
    MOV B, EBX            ; update b
    LOOP L1
    ADD EBX, 2
    MOV F, EBX            ; update f
```

Q6 Assume the following data declarations.

```
.data
   string1  byte     "FAST-National University of Computer and Emerging Sciences ("
   string2  byte     "Department of Computer Science)"
```

The requirement is to append **string2** at the end of **string1**, making **string1** a null terminated string.



string2 is stored in memory adjacent to string1 which means that only null termination is needed.