Objective: Examine the Instruction Cycle and observe the memory resident code in execution:

Follow the following steps:

i) Run a command prompt.

ii) Load the DEBUG by typing the command *debug*

iii) You will see the DEBUG CURSOR as a blinking hyphen "-"

iv) Assemble the following code into absolute memory location starting at 100 HEX. Follow exactly the screen bellow.

```
C:\Documents and Settings\USER>debug
-A100
0A9A:0100   MOV   AL,20
0A9A:0102   MOV   BL,25
0A9A:0104   ADD   AL,BL
0A9A:0106   SUB   AL,BL
0A9A:0108   MOV   AH,4C
0A9A:010A   INT   21
0A9A:010C   <just press enter key>
--
```

NOTE: Here the Segment Address starts at 0A9A:
It may be different in your PC.

Please take note for the following, for the text YOUR_ID, type in your own student ID, for example SN123456

```
-N=C:\USERS\YOUR_ID\MYINST1.COM        < RESERVES A FILE IN C:\LAB, or where ever
                                         you want it to be>
-RCX                                   < TO TELL HOW BIG THE FILE IN BYTES IS>
CX 0000                                < NOW IT IS ONLY 0 BYTE>
:50                                    < WE ENTER 50 TO SET IT FOR 50 BYTES>
-RBX                                   < SEE THE VALUE OF BASE COUNTER>
BX 0000                                < IT IS 0. DON'T CHANGE IT.>
:                                      <JUST PRESS ENTER TO LEAVE IT AT 0>
-W                                     < NOW WRITE IT ON THE FLOPPY DISK>
Writing 00050 bytes                    <IT IS WRITING IT>
-Q                                     <QUIT THE DEBUGGER>
C:\WINNT\system32>DIR
 Volume in drive A has no label.
 Volume Serial Number is 7CA5-6E80

 Directory of A:\
06/28/01  08:03p              80 MYINST1.COM    <HERE IT IS !>
               1 File(s)           80 bytes
                          1,028,096 bytes free

C:\ Documents and Settings\USER >
```

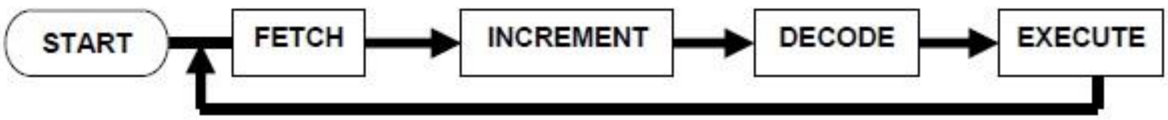
OK! now let's continue with the debugger and step through the program by inserting the break points and see the internal state of the CPU registers after execution of each instruction.

We first load our program from the previous location.

```
C:\ Documents and Settings\USER>debug C:\USERS\YOUR_ID\MYINST1.COM <load our prog.>
-U100                                                  <Unassemble the program>
0AD3:0100 B020      MOV   AL,20                        <Here it is!>
0AD3:0102 B325      MOV   BL,25
0AD3:0104 00D8      ADD   AL,BL
0AD3:0106 28D8      SUB   AL,BL
0AD3:0108 B44C      MOV   AH,4C
0AD3:010A CD21      INT   21                 <OUR PROGRAM IS UP TO HERE. You see>
                                             <some garbage following. Just ignore them>

-R
AX=0000 BX=0000 CX=0050 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0AD3 ES=0AD3 SS=0AD3 CS=0AD3 IP=0100  NV UP EI PL NZ NA PO NC
0AD3:0100 B020      MOV   AL,20
-G=100 102

AX=0020 BX=0000 CX=0050 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0AD3 ES=0AD3 SS=0AD3 CS=0AD3 IP=0102  NV UP EI PL NZ NA PO NC
0AD3:0102 B325      MOV   BL,25
-G=102 104

AX=0020 BX=0025 CX=0050 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0AD3 ES=0AD3 SS=0AD3 CS=0AD3 IP=0104  NV UP EI PL NZ NA PO NC
0AD3:0104 00D8      ADD   AL,BL
-G=104 106

AX=0045 BX=0025 CX=0050 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0AD3 ES=0AD3 SS=0AD3 CS=0AD3 IP=0106  NV UP EI PL NZ NA PO NC
0AD3:0106 28D8      SUB   AL,BL
-G=106 108

AX=0020 BX=0025 CX=0050 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0AD3 ES=0AD3 SS=0AD3 CS=0AD3 IP=0108  NV UP EI PL NZ NA PO NC
0AD3:0108 B44C      MOV   AH,4C
-G=108 10A

AX=4C20 BX=0025 CX=0050 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0AD3 ES=0AD3 SS=0AD3 CS=0AD3 IP=010A  NV UP EI PL NZ NA PO NC
0AD3:010A CD21      INT   21
-
```

Now let's examine the execution of each instruction after each break point.

Observe the value of the affected registers as the result of each instruction execution. Especially check the value of the IP (Instruction Pointer). And try to use your imagination to visualise the MEMORY RESIDENT PROGRAM EXECUTION within the context of INSTRUCTION CYCLE in a BUS ARCHITECTURED COMPUTER SYSTEM.
Try to visualise the INFINITE LOOP OF a simple Instruction Cycle:



Remember:

1. FETCH ==> Get the instruction addressed by IP in the memory and load it into MBR , thus:
   1.1 IP ==>MAR
   1.2 [MAR] ==> DATA BUS
   1.3 DATA BUS ==> MBR

2. INCREMENT ==> Increment the IP by the increment factor interpreted from the instruction's OPERATION CODE (OP CODE) to get ready for next fetch, thus:
   2.1 IP = IP + <INCREMENT FACTOR>

3. DECODE ==> Send the Instruction code in the MBR to IR so that IR interpret it and generate the EXECUTION CONTROL SIGNALS needed for its execution, thus:
   3.1 MBR ==> IR
   3.2 IR ==> <Produce the micro code needed for the execution control at the IR's output>

4. EXECUTE ==> Send the IR's Output to the Seuencer to starts the execution sequence, thus:
   4.1 IR ==> Sequencer