Laboratory 01 – x86 and C Refresher

CpE 185 / EEE 174 Lab Section 02

Vigomar Kim Algador

SUMMER 2022

TABLE OF CONTENTS

INTRODUCTION

This laboratory shows the introduction and familiarize the student about x86 with the Intel Architecture using debuggers, assemblers, and hand assembly. Different tools were being introduced that allows the students enter a program, assemble, execute, debug, and modify. These tools and methods are essential to develop as to prepare the students for upcoming future laboratories as well as to the projects and technology industries for the students were taking path on. This laboratory is divided into 3 different parts.

The first part of the laboratory will help the student familiarize to the laboratory equipment and a program in the DEBUG environment. With the given procedure for this experiment, the student must follow and understand the step-by-step process to apply in upcoming laboratories. There are different methods introduced to the student understanding each instruction and commands. Additionally, the student used knowledge on C programming on making the assembly program equivalent to the C programming.

The second part of the laboratory shows modification of the first part of the laboratory. The student must use the same techniques and knowledge from the first part of the laboratory and put changes of the same program from the first part. In addition, there are new concepts added that the student must learn and apply to the program. Same as the first part, the student wrote the equivalent C programming of the assembly program.

PART 1: INTRODUCTION TO DEBUG AND C REFRESHER

A. PRELAB

Flow chart

The student created a flowchart to understand the instructions of the program by depicting a real-life situation shown in figure 1.1.

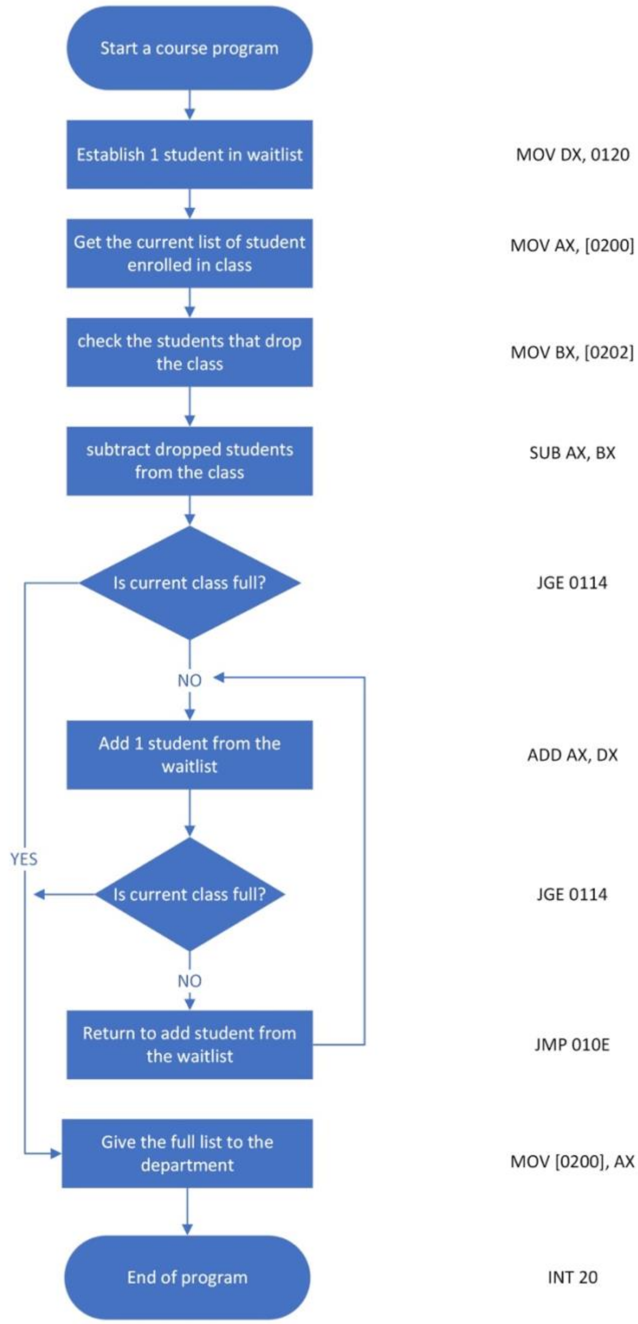


Figure 1.1. Flowchart of the debug program using a representation of a real-life situation

Hand Assembly

On the next one, the student used a hand assembly technique to understand the instruction equivalent to machine codes. Using the hand assembly instruction handout, the student matches each instruction corresponding binary and translate to hexadecimal. For example, “MOV DX, 0120” means immediate to register which the instruction format is “1011 wreg: immediate data”. Investigating each corresponding mnemonics such as w = 1, reg = 010, and data 2001h, the student was able to generate the binary: 1011 1010 200h which equivalent to BA2001 in hexadecimal. The full hand assembly is shown below:

Instruction: MOV DX, 0120

Address: CS : 100   Operation: MOV   Dest: DX   Source: 0120

		immediate to register			
Instruction Format		1011 wreg : immediate data			
		w = 1	reg = 010	data = 2001h	
Binary:	1011	1010	2001h		
	B	A	2001		
Hex:	BA2001				

Instruction: MOV AX, [0200]

Address: CS : 103   Operation: MOV   Dest: AX   Source: 0200

		memory to AX		
Instruction Format		1010 000w : full displacement		
		w= 1		
Binary:	1010	0001	0002h	
	A	1	2	
Hex:	A10002			

Instruction: MOV BX, [0202]

Address: CS : 106   Operation: MOV   Dest: BX   Source: 0202

		memory to reg			
Instruction Format		1000 101w : mod reg r/m			
		w= 1	mod = 00	reg = 011	r/m = 110
Binary:	1000	1011	0001	1110	
	8	B	1	E	
Hex:	8B1E0202				

Instruction: SUB AX, BX

Address: CS : 10A   Operation: SUB   Dest: AX   Source: BX

register1 to register2

Instruction Format   0010 100w : 11 reg1 reg2 w= 1 reg1 = 011 reg2 = 000

Binary: 0010 1001 1101 1000

2 9 D 8

Hex: 29D8

Instruction: JGE 0114

Address: CS : 10C Operation: JGE Dest: 0114 Source:

8-bit displacement

Instruction Format 0111 tttn : 8-bit displacement

tttn = 1101

Binary: 0111 1101

7 D 06h

Hex: 7D06

Instruction: ADD AX, DX

Address: CS : 10E Operation: ADD Dest: AX Source: DX

register1 to register2

Instruction Format 0000 000w : 11 reg1 reg2

w = 1 reg1 = 010 reg2 = 000

Binary: 0000 0001 1101 0000

0 1 D 0

Hex: 01D0

Instruction: JGE 0114

Address: CS : 110 Operation: JGE Dest: 0114 Source:

8-bit displacement

Instruction Format 0111 tttn : 8-bit displacement

tttn = 1101

Binary: 0111 1101

7 D 02h

Hex: 7D02

Instruction: JMP 010E

Address: CS : 112 Operation: JMP Dest: 010E Source:

short

Instruction Format 1110 1011 : 8-bit displacement

Binary: 1110 1011

E B FA

Hex: EBFA

Instruction: MOV [0200], AX

Address: CS : 114 Operation: MOV 0200 Source: AX

AX to memory

Instruction Format 1010 001w : full displacement

w = 1

Binary: 1010 0011

A 3 0002h

Hex: A30002

Instruction: INT 20

Address: CS : 117 Operation: INT Dest: 0020 Source:

INT n - Interrupt Type n

Instruction Format 1100 1101 : type

Binary: 1100 1101

C D 20h

Hex: CD20

B. INTRODUCTION TO DEBUG

Step 1:

At the beginning, the student was introduced on starting up a virtual machine called VM Workstation software using a Remote Lab Computers. After, the student was able to access that led to a debug command prompt and entered debug mode by typing “debug.” Following, the student typed “?” which listed the DEBUG commands in the window. The whole window showed in figure 1.1.

```
MS-DOS Prompt - DEBUG
-?
assemble      A [address]
compare       C range address
dump          D [range]
enter         E address [list]
fill          F range list
go            G [=address] [addresses]
hex           H value1 value2
input         I port
load          L [address] [drive] [firstsector] [number]
move          M range address
name          N [pathname] [arglist]
output        O port byte
proceed       P [=address] [number]
quit          Q
register      R [register]
search        S range list
trace         T [=address] [value]
unassemble    U [range]
write         W [address] [drive] [firstsector] [number]
allocate expanded memory        XA [#pages]
deallocate expanded memory      XD [handle]
map expanded memory pages       XM [Lpage] [Ppage] [handle]
display expanded memory status  XS
-
```

Figure 1.1 List of DEBUG commands

Step 2:

For the next step, the student used the DEBUG “dump” command by typing “d” to display the contents of the memory locations. The first command typed in the prompt is “d 100” which showed in figure 1.2. The beginning at code segment showed “0F68” and the instruction pointer showed “0100” mean it shows the current address and the offset of where it is. The center column represents the contents of the memory, and the last column represents the ASCII character of the contents.

```
-d 100
0F68:0100  DE E8 45 FA AC AA 3C 0D-75 FA 56 8B 36 92 DE 89   ..E...<.u.V.6...
0F68:0110  4C FE 5E 8E 06 08 D3 26-80 3E 43 04 34 00 57 0F   L.^....&.>C.4.W.
0F68:0120  BA 42 86 E9 65 FE BF 81-00 8B 36 92 DE 8B 44 FE   .B..e.....6...D.
0F68:0130  BE C6 DB 8B 74 09 03 C6-50 E8 0D FA 58 E8 5A 00   ....t...P...X.Z.
0F68:0140  03 F1 2B C6 8B C8 E8 7B-F4 83 F9 7F 72 0B B9 7E   ..+....{....r..~
0F68:0150  00 F3 A4 B0 0D AA 47 EB-08 AC AA 3C 0D 74 02 EB   ......G....<.t..
0F68:0160  F8 8B CF 81 E9 82 00 26-88 0E 80 00 C3 8B 1E 92   .......&........
0F68:0170  DE BE 1A D4 BA FF FF B8-00 AE CD 2F 3C 00 C3 A0   ........../<...
```

Figure 1.2. The content of the memory location at 0100.

The next command typed is “d 0100 0110” showed only the contents from the location “0100” to the location “0110” shown in figure 1.3.

```
-d 0100 0110
0F68:0100  DE E8 45 FA AC AA 3C 0D-75 FA 56 8B 36 92 DE 89   ..E...<.u.V.6...
0F68:0110  4C                                                L
```

Figure 1.3. The content of the memory location up to “0110”.

On the other hand, typing “d 0100 0120” displays the contents from location “0100” to the location “0200” shown in figure 1.4.

```
-d 0100 0200
0F68:0100  DE E8 45 FA AC AA 3C 0D-75 FA 56 8B 36 92 DE 89   ..E...<.u.V.6...
0F68:0110  4C FE 5E 8E 06 08 D3 26-80 3E 43 04 34 00 57 0F   L.^....&.>C.4.W.
0F68:0120  BA 42 86 E9 65 FE BF 81-00 8B 36 92 DE 8B 44 FE   .B..e.....6...D.
0F68:0130  BE C6 DB 8B 74 09 03 C6-50 E8 0D FA 58 E8 5A 00   ....t...P...X.Z.
0F68:0140  03 F1 2B C6 8B C8 E8 7B-F4 83 F9 7F 72 0B B9 7E   ..+....{....r..~
0F68:0150  00 F3 A4 B0 0D AA 47 EB-08 AC AA 3C 0D 74 02 EB   ......G....<.t..
0F68:0160  F8 8B CF 81 E9 82 00 26-88 0E 80 00 C3 8B 1E 92   .......&........
0F68:0170  DE BE 1A D4 BA FF FF B8-00 AE CD 2F 3C 00 C3 A0   .........../<...
0F68:0180  DB E2 0A C0 74 09 56 57-E8 2A 21 5F 5E 73 0A B9   ....t.VW.*!_^s..
0F68:0190  04 01 FC 56 57 F3 A4 5F-5E C3 50 56 33 C9 33 DB   ...VW.._^.PV3.3.
0F68:01A0  AC E8 5F 23 74 19 3C 0D-74 15 F6 C7 20 75 06 3A   .._#t.<.t... u.:
0F68:01B0  06 0C D3 74 0A 41 3C 22-75 E6 80 F7 20 EB E1 5E   ...t.A<"u... ..^
0F68:01C0  58 C3 A1 E1 D7 8B 36 E3-D7 C6 06 25 D9 00 C6 06   X.....6....%....
0F68:01D0  21 D9 00 8B 36 E3 D7 8B-0E E1 D7 8B D6 E3 42 51   !...6.........BQ
0F68:01E0  56 5B 2B DE 59 03 CB 8B-D6 C6 06 C5 DB 00 E3 31   V[+.Y..........1
0F68:01F0  49 AC E8 D9 F6 74 08 49-46 FE 06 C5 DB EB EF E8   I....t.IF.......
0F68:0200  DB                                                .
```

Figure 1.4 The content of the memory location up to “0200”.

Step 3:

For the next step, the student uses DEBUG “enter” command by typing “e” in the prompt. The student typed “e100” which means the memory location will start at 0100 and the values are written in hexadecimal. On the other hand, the student entered with the given machine codes shown in the figure 1.5.

```
-e100
0F68:0100  DE.ba   E8.20   45.01   FA.a1   AC.00   AA.02   3C.8b   0D.1e
0F68:0108  75.02   FA.02   56.29   8B.d8   36.7d   92.06   DE.01   89.d0
0F68:0110  4C.7d   FE.02   5E.eb   8E.fa   06.a3   08.00   D3.02   26.cd
0F68:0118  80.20
```

Figure 1.5. The machine code written in hexadecimal values starting at memory location “100”.

Step 4:

After entering the machine codes, the student used the DEBUG “unassemble” command by typing “u” to see the whole program. The student entered “u100 118” which shows the program listing from memory location “100” to “118” shown in figure 1.6. This shows the equivalent mnemonics of machine codes. For example, the machine code BA2001 equivalents to MOV DX, 0120.

```
-u100 118
0F68:0100 BA2001        MOV     DX,0120
0F68:0103 A10002        MOV     AX,[0200]
0F68:0106 8B1E0202      MOV     BX,[0202]
0F68:010A 29D8          SUB     AX,BX
0F68:010C 7D06          JGE     0114
0F68:010E 01D0          ADD     AX,DX
0F68:0110 7D02          JGE     0114
0F68:0112 EBFA          JMP     010E
0F68:0114 A30002        MOV     [0200],AX
0F68:0117 CD20          INT     20
```

Figure 1.6. The list of unassembled from memory location “0100” to “0118”.

Looking at the right 2 columns, it shows the clear instruction for each memory location. From the first one, it shows “MOV DX, 0120” meaning that the value “0120” copies to “DX” registry. On the other hand, “MOV AX, [0200]” means that the value in the address “0200” copies to “AX” registry and the bracket has its significant role to it. Other instructions such as “SUB” and “ADD” means subtract and add. For example, “SUB AX, BX” means AX registry is being subtracted by BX and the result value will be stored in AX. Other than that, “JGE” instruction stands for “jump greater or equal to”, for example, “JGE 0114” meaning jump to memory location “0114” if AX registry is greater than or equal to 0. Other similar instruction will be “JMP” which means “unconditional jump” to a given memory location. Lastly, the “INT” instruction indicates interrupt which “INT 20” makes the program end. This is very important as for the last instruction to end the program

Step 5:

The student used the DEBUG “register modify” command by typing “r” which showed the registers shown in figure 1.7. This shows all the values stored in the register.

```
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0100   NV UP EI PL NZ NA PO NC
0F68:0100 BA2001        MOV     DX,0120
```

Figure 1.7. register information stored in memory location “0100”.

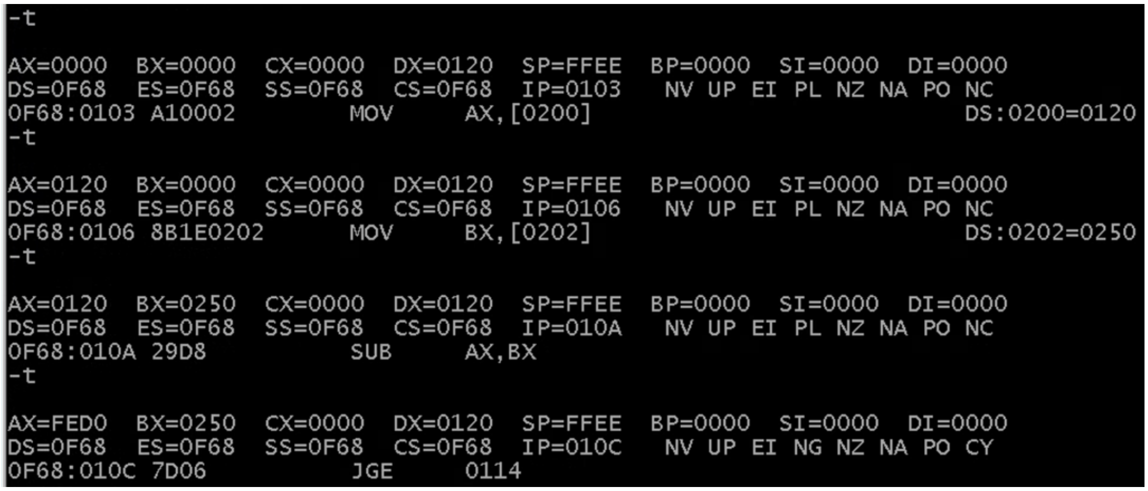
Step 6:

After that, the student used the DEBUG “trace” command. At first, the students needed to setup the values in memory by typing “e200” with the given values and display the memory values by typing “d200 203” shown in figure 1.8.

```
-e200
0F68:0200  DB.20   F9.01   75.50   04.02
-d200 203
0F68:0200  20 01 50 02                                   .P.
-t
```

Figure 1.8. setting values in the memory “0200” and display the memory values.

After entering the values, the next one needed is by using “trace” command in which the student type “t” in the command. This command needs to run a few times until it hits the “INT 20H” instruction. The snippet of using "trace” command in the prompt in figure 1.9. The whole tracing chart for the first run is shown in Table 1.1.


```
-t

AX=0000  BX=0000  CX=0000  DX=0120  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0103   NV UP EI PL NZ NA PO NC
0F68:0103 A10002        MOV     AX,[0200]                        DS:0200=0120
-t

AX=0120  BX=0000  CX=0000  DX=0120  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0106   NV UP EI PL NZ NA PO NC
0F68:0106 8B1E0202      MOV     BX,[0202]                        DS:0202=0250
-t

AX=0120  BX=0250  CX=0000  DX=0120  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=010A   NV UP EI PL NZ NA PO NC
0F68:010A 29D8          SUB     AX,BX
-t

AX=FED0  BX=0250  CX=0000  DX=0120  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=010C   NV UP EI NG NZ NA PO CY
0F68:010C 7D06          JGE     0114
```

Figure 1.9. snippet of trace command in four memory locations: 0103, 0106, 010A, 010C

Table 1.1. Tracing chart for the first run

AX:	BX:	CX:	DX:	OF:	ZF:	SF:	CS:	IP:	DS:200	DS:202	Next Instruction:
0000	0000	0000	0000	NV (0)	NZ (0)	PL (0)	0F68	0100	0120	0250	Mov DX, 120
0000	0000	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	0103	0120	0250	Mov AX [0200]
0120	0000	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	0106	0120	0250	Mov BX [0202]
0120	0250	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	010A	0120	0250	SUB AX, BX
FED0	0250	0000	0120	NV (0)	NZ (0)	NG (1)	0F68	010C	0120	0250	JGE 0114
FED0	0250	0000	0120	NV (0)	NZ (0)	NG (1)	0F68	010E	0120	0250	ADD AX, DX
FFF0	0250	0000	0120	NV (0)	NZ (0)	NG (1)	0F68	0110	0120	0250	JGE 0114
FFF0	0250	0000	0120	NV (0)	NZ (0)	NG (1)	0F68	0112	0120	0250	JMP 010E
FFF0	0250	0000	0120	NV (0)	NZ (0)	NG (1)	0F68	010E	0120	0250	ADD AX, DX
0110	0250	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	0110	0120	0250	JGE 0114
0110	0250	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	0114	0120	0250	MOV [0200], AX
0110	0250	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	0117	0110	0250	INT 20

Step 7:

In the last step, the student is now finally run the entire program using the DEBUG “Go” command. In this step, there are multiple ways that the command “go” can be executed such as starting at the existing location of the IP, specified IP, or previous methods combined with setting of “break points.” In figure 1.10, the student showed different ways. For example, entering “g” using the command when the IP has been set; Entering “g=100” will start at memory location 100; and “g=100 10E” sets a breakpoint.

```
-rip
IP 0117
:100
-G

Program terminated normally
-G=100

Program terminated normally
-G=100 10E

AX=FEA0  BX=0250  CX=0000  DX=0120  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=010E   NV UP EI NG NZ NA PE CY
0F68:010E 01D0          ADD     AX,DX
```

Figure 1.10. Using the “g” command in different methods.

C. C PROGRAMMING REFRESHER

In this part of the laboratory, the students are required to write a C program. The first task required is to write a program that outputs "Hello" while the second task requires to add two numbers. The first and second task are written in one program shown in figure 1.11.

```c
// Vigomar Kim Algador
// CPE185 / EEE174   : SECTION 02
// LABORATORY 01 PART 01
#include <stdio.h>

int main() {
    printf("Hello\n");      // output Hello

    int a = 10, b = 20;     // initialize a = 10 and b = 20

    printf ("a=%d, b=%d\n",a,b);  // print the values a, b

    int sum = a + b;        // add a and b store in sum

    printf ("the sum is %d\n", sum);    // print the sum
    return 0;
}
```

```
> clang-7 -pthread -lm -o main main.c
> ./main
Hello
a=10, b=20
the sum is 30
>
```

Figure 1.11. C programming code (left) and output (right).

For the final task, the student is required to write and perform the same function as the assembly language program used in Debug. C programming seems a little complicated, however, it makes things and writing easier than assembly program. Experiencing on writing in assembly seems to be tedious rather than flexible unlike high-programming languages like C. However, Assembly is handy in terms of specifying instructions and executes quickly. Pointing out the difference in the program shown in figure 1.12, instead of using jump statement in Assembly, the student used while loop.

```c
// Vigomar Kim Algador
// CPE185 / EEE174   : SECTION 02
// LABORATORY 01 PART 01
#include <stdio.h>

int main() {
    int memory0200 = 0x0120, memory0202 = 0x0250;
    int AX, BX, DX;

    DX = 0x0120;            // MOV DX, 0120
    AX = memory0200;        // MOV AX, [0200]
    BX = memory0202;        // MOV BX, [0202]
    AX -= BX;               // SUB AX, BX

    // 010E location
    while (AX < 0)          // while the value of AX < 0 loop
      AX += DX;             // ADD AX, DX

    // 0114 location
    memory0200 = AX;        // MOV [0200], AX

    printf("AX = %x, BX = %x, DX = %x\n", AX, BX, DX);
    printf("memory0200 = %x, memory0202 = %x\n",
memory0200, memory0202);
    return 0;               // INT 20 (exit code)
}
```

```
> clang-7 -pthread -lm -o main main.c
> ./main
AX = 110, BX = 250, DX = 120
memory0200 = 110, memory0202 = 250
>
```

Figure 1.12. The same process for assembly written C programming

PART 2: HAND ASSEMBLY AND C PROGRAMMING

In this part of the laboratory, the students were introduced to develop an 8-bit version of the previous part of this laboratory. The student was given different specification to modify the program. The first one to modify is the student needs to use only one JGE instruction. Another modification is that the student doesn't allow to use a specific register and only use consecutive memory locations for data starting at the given address from the laboratory instructor. The instruction for using registers and memory locations depend on the student's first and last name. Since the student's first name starts with "V" and last name starts with "A", the restriction for using register will be "A" and the address to start with is "454". The student was able to modify the flowchart from the first part with the new registers, locations and using only one JGE instructions. Here is the flowchart modified shown in figure 2.1.

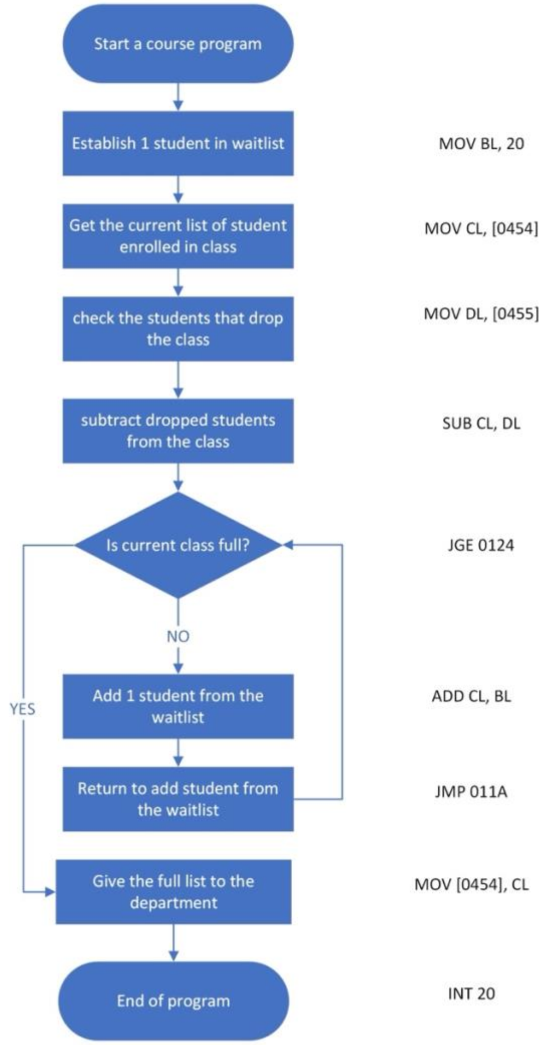


Figure 2.1. Flowchart representation of the program with modification from part 1

Other than that, the student must display the name and title of the program and track of how many times for the loop done. With all of these modifications needed, the student was able to create an outline of the whole assembly program shown in figure 2.1. As you see, we added more instructions such as “MOV DX, 0460”, “MOV AH, 09”, and “INT 21” in which helps to print out to the screen, and “INC BYTE PTR [0456]” in which helps to count the number of loops in the program.

```
MOV     DX,0460
MOV     AH,09
INT     21
MOV     DX,0490
MOV     AH,09
INT     21
MOV     BL,20
MOV     CL,[0454]
MOV     DL,[0455]
SUB     CL,DL
JGE     0124
INC     BYTE PTR [0456]
ADD     CL,BL
JMP     011A
MOV     [0454],CL
MOV     DX,0456
MOV     AH,09
INT     21
INT     20
```

Figure 2.2. The full assembly language outline of the program

For the next one, the student needed to do hand assembly in able to obtain the hexadecimal equivalent for each instruction to input in the prompt. The snippet of the hand assembly is shown in figure 2.2. where it shows the instruction “MOV BL, 20”, “Mov CL, [0454]”, and “MOV DL, [0455]”.

```
Instruction:  MOV BL, 20
Address:      CS : 10E          Operation: MOV     Dest: BL     Source: 20
                   immediate to register
  Instruction Format   1011 wreg : immediate data
                          w = 0      reg = 011   data = 20h
Binary:          1011        0011        20h
                    B           3        20h
Hex:          B320

Instruction:  MOV CL, [0454]
Address:      CS : 110          Operation: MOV     Dest: CL     Source: 0454
                   memory to reg
  Instruction Format   1000 101w : mod reg r/m
                          w= 0     mod = 00     reg = 001     r/m = 110
Binary:          1000        1010        0000        1110        5404h
                    8           A           0           E         5404
Hex:          8A0E5404

Instruction:  MOV DL, [0455]
Address:      CS : 114          Operation: MOV     Dest: DL     Source: 0455
                   memory to reg
  Instruction Format   1000 101w : mod reg r/m
                   w= 0            mod = 00     reg = 010     r/m = 110
Binary:          1000        1010        0001        0110        5504h
                    8           A           1           6         5504
Hex:          8A165504
```

Figure 2.3. Snippet of hand assembly of the program.

After obtaining the corresponding hexadecimal, the student uses the DOS prompt to input the whole program. The unassembled form shown in figure 2.3. Comparing the entered program to the hand assembly, the student confirmed there are no errors with the instructions.

```
-u100 12F
0F68:0100 BA6004        MOV     DX,0460
0F68:0103 B409          MOV     AH,09
0F68:0105 CD21          INT     21
0F68:0107 BA9004        MOV     DX,0490
0F68:010A B409          MOV     AH,09
0F68:010C CD21          INT     21
0F68:010E B320          MOV     BL,20
0F68:0110 8A0E5404      MOV     CL,[0454]
0F68:0114 8A165504      MOV     DL,[0455]
0F68:0118 28D1          SUB     CL,DL
0F68:011A 7D08          JGE     0124
0F68:011C FE065604      INC     BYTE PTR [0456]
0F68:0120 00D9          ADD     CL,BL
0F68:0122 EBF6          JMP     011A
0F68:0124 880E5404      MOV     [0454],CL
0F68:0128 BA5604        MOV     DX,0456
0F68:012B B409          MOV     AH,09
0F68:012D CD21          INT     21
0F68:012F CD20          INT     20
```

Figure 2.4. List of unassembled of the whole program

After that, the student needed to trace the whole program for each instruction. The student was instructed to start tracing from the instruction “MOV BL, 20”. The student must do the trace for 2 runs. The first run of the tracing chart is shown in table 2.1.

Table 2.1. Tracing chart for the first run

AX:	BX:	CX:	DX:	OF:	ZF:	SF:	CS:	IP:	DS:454	DS:455	Next Instruction:
0924	0000	0000	0490	NV (0)	NZ (0)	PL (0)	0F68	010E	05	09	MOV BL, 20
0924	0020	0000	0490	NV (0)	NZ (0)	PL (0)	0F68	0110	05	09	MOV CL, [0454]
0924	0020	0005	0490	NV (0)	NZ (0)	PL (0)	0F68	0114	05	09	MOV DL, [0455]
0924	0020	0005	0409	NV (0)	NZ (0)	PL (0)	0F68	0118	05	09	SUB CL, DL
0924	0020	00FC	0409	NV (0)	NZ (0)	NG (1)	0F68	011A	05	09	JGE 0124
0924	0020	00FC	0409	NV (0)	NZ (0)	NG (1)	0F68	011C	05	09	INC BYTE PTR [0456]
0924	0020	00FC	0409	NV (0)	NZ (0)	PL (0)	0F68	0120	05	09	ADD CL, BL
0924	0020	001C	0409	NV (0)	NZ (0)	PL (0)	0F68	0122	05	09	JMP 011A
0924	0020	001C	0409	NV (0)	NZ (0)	PL (0)	0F68	011A	05	09	JGE 0124
0924	0020	001C	0409	NV (0)	NZ (0)	PL (0)	0F68	0124	05	09	MOV [0454], CL
0924	0020	001C	0409	NV (0)	NZ (0)	PL (0)	0F68	0128	1C	09	MOV DX, 0456
0924	0020	001C	0456	NV (0)	NZ (0)	PL (0)	0F68	012B	1C	09	MOV AH, 09
0924	0020	001C	0456	NV (0)	NZ (0)	PL (0)	0F68	012D	1C	09	INT 21
0924	0020	001C	0456	NV (0)	NZ (0)	PL (0)	0F68	012F	1C	09	INT 20

At the end, the student was able to run the program and collect the whole data. Here is the whole process of the program in the DOS prompt shown in figure 2.1.

```
C:\WINDOWS>debug
-E100
0F68:0100  DE.BA   E8.60   45.04   FA.B4   AC.09   AA.CD   3C.21   0D.BA
0F68:0108  75.90   FA.04   56.B4   8B.09   36.CD   92.21   DE.B3   89.20
0F68:0110  4C.8A   FE.0E   5E.54   8E.04   06.8A   08.16   D3.55   26.04
0F68:0118  80.28   3E.D1   43.7D   04.08   34.FE   00.06   57.56   0F.04
0F68:0120  BA.00   42.D9   86.EB   E9.F6   65.88   FE.0E   BF.54   81.04
0F68:0128  00.BA   8B.56   36.04   92.B4   DE.09   8B.CD   44.21   FE.CD
0F68:0130  BE.20
-
// message data for the name, 0d & 0a - cr and lf, $ - end of string
-E460 "Hello, My name is Vigomar Kim Algador" 0d 0a "$"
-
// message data for the title, 0d & 0a - cr and lf, $ - end of string
-E490 "Welcome to EEE174 / CPE185-LAB01 PART02" 0d 0a "$"
-
// data for looping, 30 - ASCII starts number, 0d & 0a - cr and lf, $ - end of string
-E456 30 0d 0a "$"
-

// input data for memory [0454] = 05 and [0455] = 09
-E454
0F68:0454  D9.05   E2.09
-
-u100 12F
0F68:0100 BA6004        MOV     DX,0460          // load DX with the value of location 0460
0F68:0103 B409          MOV     AH,09            // set the BIOS service to display the message
0F68:0105 CD21          INT     21               // DOS interrupt to display message
0F68:0107 BA9004        MOV     DX,0490          // load DX with the value of location 0490
0F68:010A B409          MOV     AH,09            // set the BIOS service to display the message
0F68:010C CD21          INT     21               // DOS interrupt to display message
0F68:010E B320          MOV     BL,20            // move 20 to BL
0F68:0110 8A0E5404      MOV     CL,[0454]        // move the value from the address [0454] to CL
0F68:0114 8A165504      MOV     DL,[0455]        // move the value from the address [0455] to DL
0F68:0118 28D1          SUB     CL,DL            // Subtract DL to CL and store the result to CL
0F68:011A 7D08          JGE     0124             // condition if CL ≥ 0 then jump to location 0124
0F68:011C FE065604      INC     BYTE PTR [0456]  // use to count the number of loops and store to memory [0456]
0F68:0120 00D9          ADD     CL,BL            // add CL and BL and store the result to CL
0F68:0122 EBF6          JMP     011A             // unconditional loop back to address 011A
0F68:0124 880E5404      MOV     [0454],CL        // move the value from CL to memory [0454]
0F68:0128 BA5604        MOV     DX,0456          // load DX with the value of location 0456
0F68:012B B409          MOV     AH,09            // set the BIOS service to display the message
0F68:012D CD21          INT     21               // DOS interrupt to display message
0F68:012F CD20          INT     20               // end of program
-
// first run and the output
-g=100
Hello, My name is Vigomar Kim Algador
Welcome to EEE174 / CPE185-LAB01 PART02
1

Program terminated normally
-
-d454 455
0F68:0450              1C 09
-
// second run and output
-g=100
Hello, My name is Vigomar Kim Algador
Welcome to EEE174 / CPE185-LAB01 PART02
1

Program terminated normally
-
-d454 455
0F68:0450              13 09
-
```

Figure 2.5. The whole process of the program in DOS prompt with comments

Addition to this part of the laboratory, the student was able to transform the assembly language to a C programming language using the knowledge and skills learned from previous classes.

main.c ×

```c
// Vigomar Kim Algador
// CPE185 / EEE174 : SECTION 02
// LABORATORY 01 PART 02
#include <stdio.h>

int main() {
  char memory0460[50] = "Hello, My name is Vigomar Kim Algador";
  char memory0490[50] = "Welcome to EEE174 / CPE185-LAB01 PART02";
  // MOV DX, 0460
  // MOV AH, 09
  // INT 21
  printf("%s\n", memory0460);

  // MOV DX, 0490
  // MOV AH, 09
  // INT 21
  printf("%s\n", memory0490);

  int memory0454 = 0x05, memory0455 = 0x09;
  int BL, CL, DL;
  int memory0456 = 0;

  BL = 0x20;      // MOV BL, 20
  CL = memory0454; // MOV CL, [0454]
  DL = memory0455; // MOV DL, [0455]
  CL -= DL;     // SUB CL, DL

  while (CL < 0) {  // JGE 011A
    memory0456++; // INC BYTE PTR [0456]
    CL += BL;  // ADD CL, BL
  }

  memory0454 = CL; // MOV [0454], CL

  // MOV DX, 0456
  // MOV AH, 09
  // INT 21
  printf("memory0456 = %d\n", memory0456);

  // printing out values of registers and locations
  printf("BL = %x, CL = %x DL = %x\n", BL, CL, DL);
  printf("memory0454 = %x, memory0455 = %x\n", memory0454, memory0455);
  return 0; // INT 20
}
```

Console Shell

```
> clang-7 -pthread -lm -o main main.c
> ./main
Hello, My name is Vigomar Kim Algador
Welcome to EEE174 / CPE185-LAB01 PART02
memory0456 = 1
BL = 20, CL = 1c DL = 9
memory0454 = 1c, memory0455 = 9
>
```

Figure 2.6. C programming of the assembly program (left) and the output (right)

CONCLUSION

In this laboratory, the student learned the process of writing in machine language. This introduced a lot of materials including the use of MS-DOS prompt, spreadsheets, and hand calculations. One of the important things is understanding the process of hand assembly of instructions and converting to input in the prompt. The first part of the laboratory really helps the student on being introduced to the new material with the step-by-step process of the use of different commands. The student was able to experiment the commands and use them in different ways. Additionally, the student was also introduced to the hand assembly and tracing chart.

On the other hand, the second part of the laboratory is where the student modified things from the first part. This introduced on converting the 16-bit program into an 8-bit program. This includes the registers, memory locations, and data within the program. This part shows the important process of planning and creating an outline before entering to the MS-DOS prompt. Listing the instructions that will be used and applying to hand assembly is an important first step. Moving to debug, the use of tracing chart is another important method to list all the registers, flags, and other things to keep track of the information of each instruction as the program runs. At the end, the student successfully to get a good result of the whole program.