MARIO PALACIOS
LAB COURSE: CpE 185 – SECTION 03
MONDAY (6:30PM – 9:10PM)
LAB 01: x86 AND C REFRESHER
INSTRUCTOR: SEAN KENNEDY

Introduction:

I will be demoing part 3 of this lab.

The x86 the most common used microprocessor and it is In Windows and Macintosh personal computers. In this lab we will become familiar with the Intel Architecture using debuggers, assemblers, un-assemblers, and hand assembly. These tools will allow us to enter programs, assemble, execute, debug, and modify programs. While also providing a refresh in C programming knowledge.

Part1. Introduction to Debug and C Refresher

Description: We will be familiarizing ourselves with the DEBUG environment, and explore the nature a program using this program.

Engineering Data:

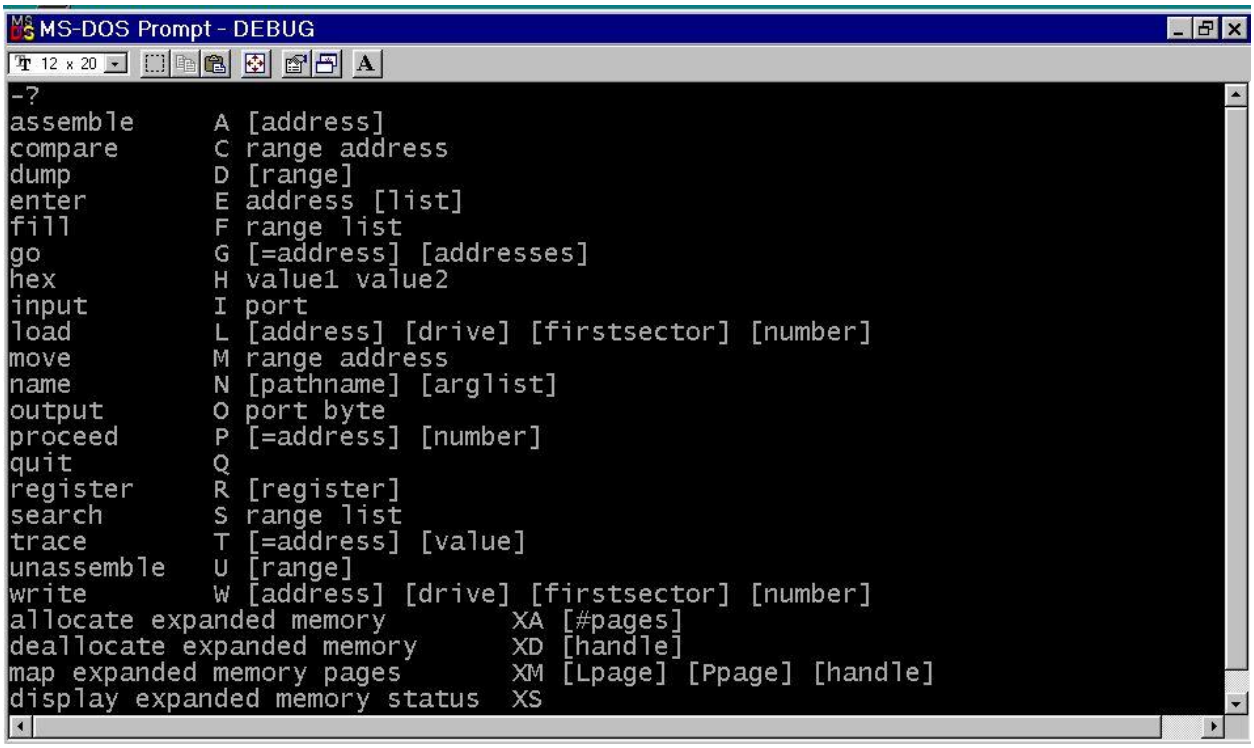

Figure 1. Typing in "?" to Show List of DEBUG Commands (Step 1)

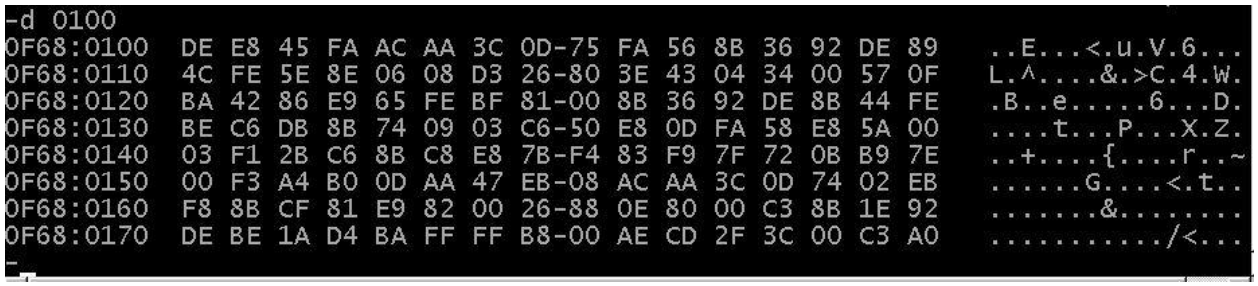

Figure 2a. "dump" Command for d 0100, Range of all Data in 100 (Step 2)

```
-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 2b. "dump" Command for d 0100 0110, Range of all Data in 100 and 110 (Step 2)

```
-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 2b. "dump" Command for d 0100 0200, Range of all Data in 100 and 200 (Step 2)

```
-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 3. Entering Assembly for Code Segment 100 (Step 3)

```
-u100 118
0F68:0100 BA2002        MOV     DX,0220
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 4. "Unassemble" Command to View Program that was Entered (Step 4)

```
-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 5. "Register Modify" to Set Instruction Pointer to Location CS: 0100 (Step 5)

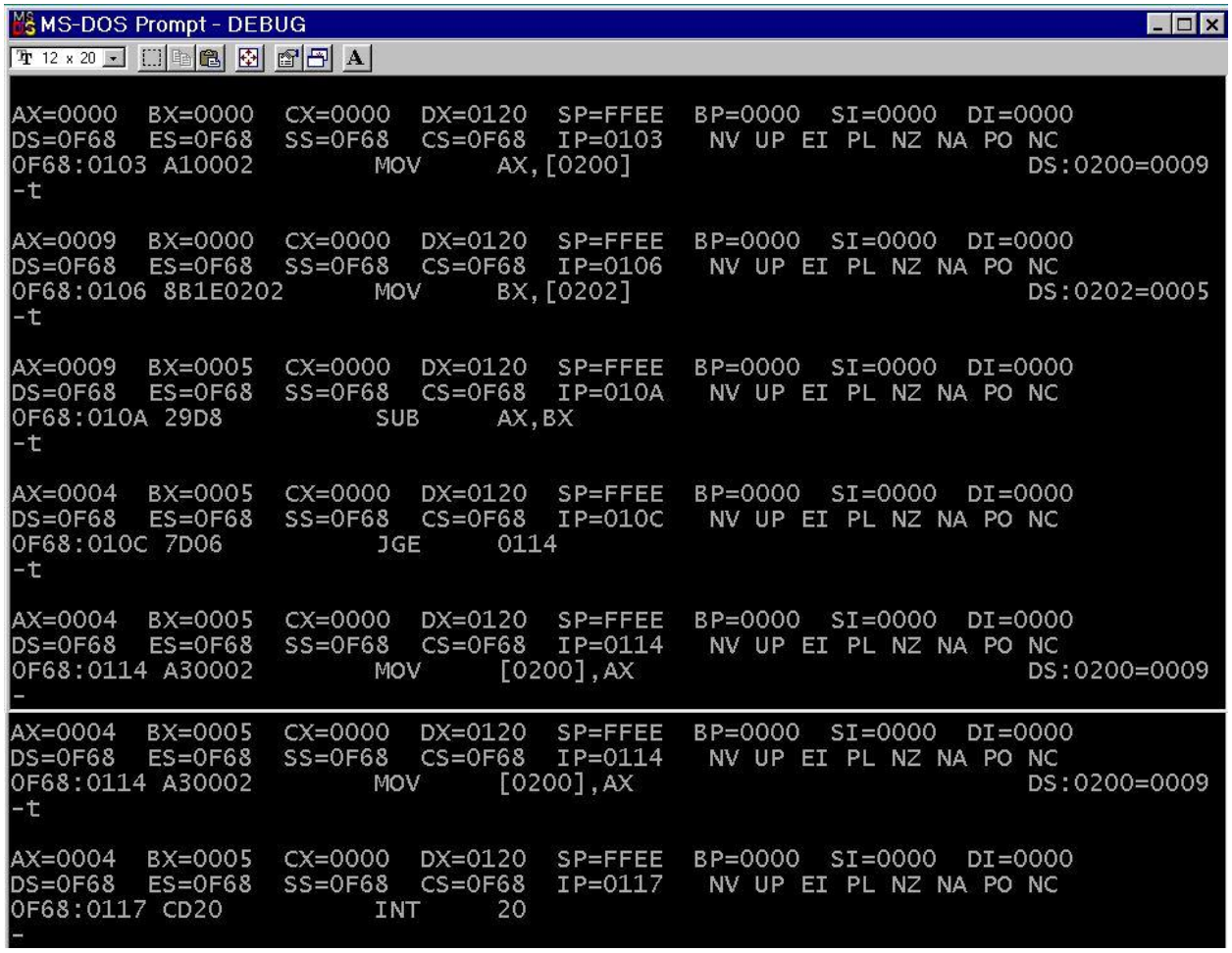

Figure 6-1. Tracing Through Program Until INT 21 is Reached

```
MS-DOS Prompt - DEBUG

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=0005
-t

AX=0005  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=0009
-t

AX=0005  BX=0009  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=FFFC  BX=0009  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 AC PE CY
0F68:010C 7D06          JGE     0114
-t

AX=FFFC  BX=0009  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 AC PE CY
0F68:010E 01D0          ADD     AX,DX
-

AX=011C  BX=0009  CX=0000  DX=0120  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0110   NV UP EI PL NZ NA PO CY
0F68:0110 7D02          JGE     0114
-t

AX=011C  BX=0009  CX=0000  DX=0120  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0114   NV UP EI PL NZ NA PO CY
0F68:0114 A30002        MOV     [0200],AX                        DS:0200=0005
-t

AX=011C  BX=0009  CX=0000  DX=0120  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0117   NV UP EI PL NZ NA PO CY
0F68:0117 CD20          INT     20
-
```

Figure 6-2. Tracing Through Program and Entering Loop

```
-rip
IP 0106
:100
-g

Program terminated normally
-g=100

Program terminated normally
-g=100 10C

AX=011A  BX=0005  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 PL NZ NA PO NC
0F68:010C 7D06          JGE     0114
-
```

Figure 7. Specifying Breakpoints using the "Go" Command

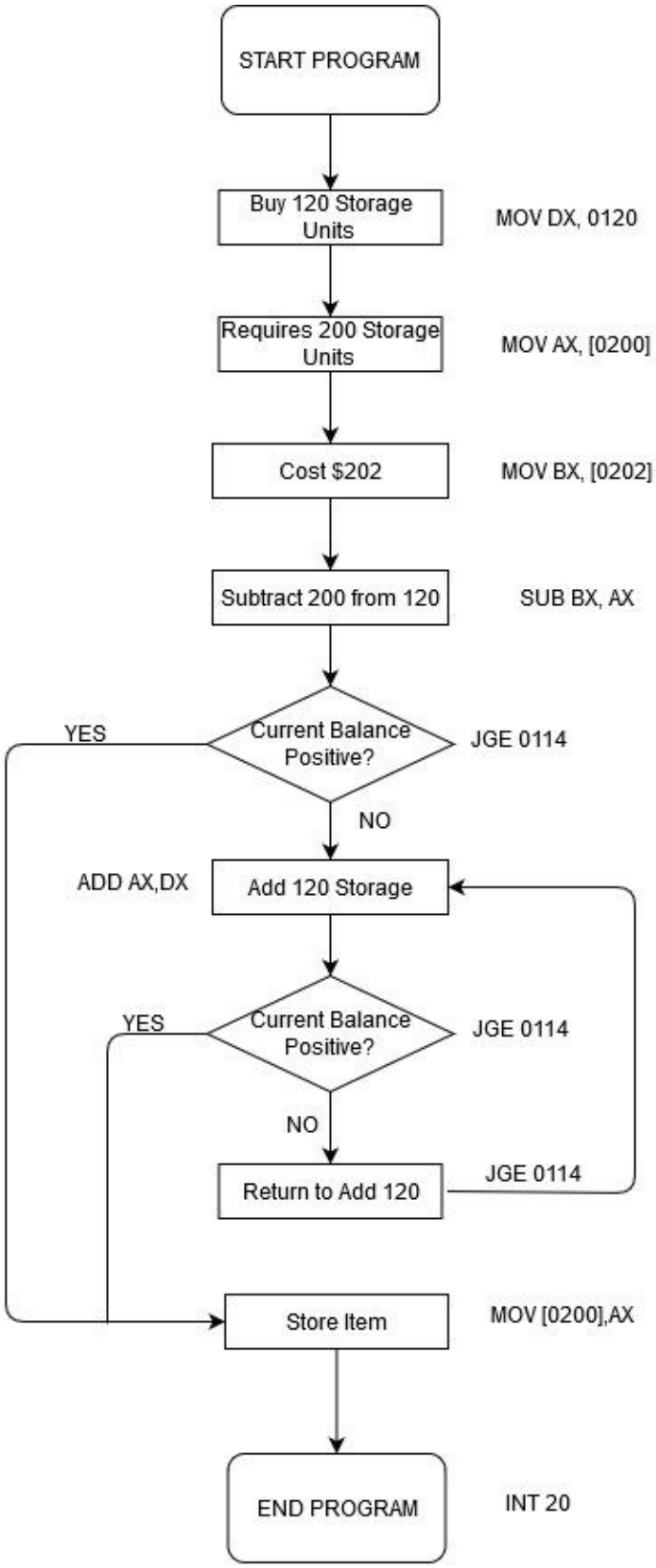

Figure 8. Flow Chart with Application of Program

Instruction MOV DX, 0120

Address: CS **100** Operation: MOV Dest.: DX Source: **120**
Immediate to reg
Instruction Format 1011 wreg immediate data
w- 1 reg- 010 immediate data- 2001h
Binary: 1011 1010 2001h
B A 2001
Hex: BA2001

Instruction MOV AX, [0200]

Address: CS **103** Operation: MOV Dest.: AX Source: **200**
memory to AX
Instruction Format 1010 000w full displacement
w- 1
Binary: 1010 0001 0002h
A 1 2
Hex: A1002

Instruction MOV BX, [0202]

Address: CS **106** Operation: MOV Dest.: BX Source: **202**

Instruction Format 1000 101w mod reg r/m
w- 1 mod- 00 reg- 011 r/m- 110
Binary: 1000 1011 0001 1110 0202h
8 B 1 E 202
Hex: 8B1E0202

Instruction SUB AX, BX

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

Instruction Format 0010 100w: 11 reg1 reg2
w-1 reg 1- 011 reg2- 000
Binary: 0010 1001 1101 1000
2 9 D
Hex: 29D8

Instruction JGE 0114

Address: CS **10C** Operation: JGE Dest.: 114 Source:

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**

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.: 114 Source:

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:

Instruction Format 1110 1011: 8-bit displacement

Binary: 1110 1011 1111 1010
E B FA
Hex: EBFA

Instruction MOV [0200], AX

Address: CS **114** Operation: MOV Dest.: 200 Source: **AX**
mem to AX
Instruction Format 1010 001w: full displacement
w- 1
Binary: 1010 0011 0002h
A 3 002h
Hex: A30002

Instruction INT 20

Address: CS **117** Operation: INT Dest.: 20 Source:
INT n
Instruction Format 1100 1101: type
type- 20
Binary: 1100 1101 0010 1000
C D 20
Hex: CD20

Figure 9. Hand Assembly Chart For Assembly Program (Above)

CpE 185												
Laboratory Exercise #1									**Name:**	Mario Palacios		
Program Tracing Chart												
	Registers:											
	AX:	**BX:**	**CX:**	**DX:**	**OF:**	**ZF:**	**SF:**	**CS:**	**IP:**	**DS:200**	**DS:202**	**Next Instruction:**
Value:--->	0000	0000	0000	0120	NV(0)	NZ(0)	PL(0)	0F68	0103	0009	0005	MOV AX, [0200]
	0009	0000	0000	0120	NV(0)	NZ(0)	PL(0)	0F68	0106	0009	0005	MOV BX, [0202]
	0009	0005	0000	0120	NV(0)	NZ(0)	PL(0)	0F68	010A	0009	0005	SUB AX, BX
	0004	0005	0000	0120	NV(0)	NZ(0)	PL(0)	0F68	010C	0009	0005	JGE 0114
	0004	0005	0000	0120	NV(0)	NZ(0)	PL(0)	0F68	0114	0009	0005	MOV [0200], AX
	0004	0005	0000	0120	NV(0)	NZ(0)	PL(0)	0F68	0117	0009	0005	INT 20
Second Run:												
	0000	0000	0000	0120	NV(0)	NZ(0)	PL(0)	0F68	0103	0005	0009	MOV AX, [0200]
	0005	0000	0000	0120	NV(0)	NZ(0)	PL(0)	0F68	0106	0005	0009	MOV BX, [0202]
	0005	0009	0000	0120	NV(0)	NZ(0)	PL(0)	0F68	010A	0005	0009	SUB AX, BX
	FFFC	0009	0000	0120	NV(0)	NZ(0)	NG(1)	0F68	010C	0005	0009	JGE 0114
	FFFC	0009	0000	0120	NV(0)	NZ(0)	NG(1)	0F68	010E	0005	0009	ADD AX, DX
	011C	0009	0000	0120	NV(0)	NZ(0)	PL(0)	0F68	0110	0005	0009	JGE 0114
	011C	0009	0000	0120	NV(0)	NZ(0)	PL(0)	0F68	0114	0005	0009	MOV [0200], AX
	011C	0009	0000	0120	NV(0)	NZ(0)	PL(0)	0F68	0117	0005	0009	INT 20

Figure 10. Tracing Chart with Two Runs with One Going Into The Loop

```
#include <stdio.h>
#include <stdlib.h>

int main (void)
{
        int a = 3;
        int b = 5;
        int sum = a + b;
        printf ("Hello\n");
        printf ("The sum of a + b = %i\n\n\n", sum);

        int dx = 120;
        int ax,bx;
        scanf ("%d", &ax);
        scanf ("%d", &bx);
        ax = bx - ax;
        while(ax <= 0)
        {
    printf ("%i\n", ax);
                ax = ax + dx;
        }
        printf ("The new value of ax is: %i", ax);
        return  EXIT_SUCCESS;
}
```

Figure 11. C Programming Refresher

Conclusion: This part of the lab exposed me to how Assembly programming is essential to moving data around in a microprocessor. Using the DEBUG environment was intriguing because of all the commands it can execute. It also served as a good refresher on C programming.

Part 2.Hand Assembly and C Programming

Description: The goal of this part is to develop an 8-bit version of the program done in the previous part, while only using one JGE instruction, and no use of the CX register.

Engineering Data:

```
0F68:0100 BA0002      MOV   DX, 0200
0F68:0103 B409        MOV   AH, 09
0F68:0105 CD21        INT   21
0F68:0107 B250        MOV   DL, 50
0F68:0109 A05403      MOV   AL, [0354]
0F68:010C 8A1E5503    MOV   BL, [0355]
0F68:0110 28D8        SUB   AL, BL
0F68:0112 FE065803    INC   BYTE PTR [0358]
0F68:0116 7D04        JGE   011C
0F68:0118 00D0        ADD   AL, DL
0F68:011A EBFA        JMP   0116
0F68:011C A25403      MOV   [0354], AL
0F68:011F BA5803      MOV   DX, 0358
0F68:0122 B409        MOV   AH, 09
0F68:0124 CD21        INT   21
0F68:0126 CD20        INT   20
```

Figure 12. Final Hand Assembly Program with Title and Counter

	Registers:											
	AL:	BL:	CX	DL:	OF:	ZF:	SF:	CS:	IP:	DS:0354	DS:0355	Next Instruction:
Value:--->	0000	0000	Not Used	0050	NV(0)	NZ(0)	PL(0)	0F68	0102	50	65	MOV AL, [0354]
	0050	0000	Not Used	0050	NV(0)	NZ(0)	PL(0)	0F68	0105	50	65	MOV BL, [0355]
	0050	0065	Not Used	0050	NV(0)	NZ(0)	PL(0)	0F68	0109	50	65	SUB AL, BL
	00EB	0065	Not Used	0050	NV(0)	NZ(0)	NG(1)	0F68	010B	50	65	JGE 0111
	00EB	0065	Not Used	0050	NV(0)	NZ(0)	PL(0)	0F68	010D	50	65	ADD AL, DL
	003B	0065	Not Used	0050	NV(0)	NZ(0)	PL(0)	0F68	010F	50	65	JMP 010B
	003B	0065	Not Used	0050	NV(0)	NZ(0)	PL(0)	0F68	010B	50	65	JGE 0111
	003B	0065	Not Used	0050	NV(0)	NZ(0)	PL(0)	0F68	0111	50	65	MOV [0354], AL
	003B	0065	Not Used	0050	NV(0)	NZ(0)	PL(0)	0F68	0114	50	65	INT 20
Second Run												
	0000	0000	Not Used	0050	NV(0)	NZ(0)	PL(0)	0F68	0102	75	90	MOV AL, [0354]
	0075	0000	Not Used	0050	NV(0)	NZ(0)	PL(0)	0F68	0105	75	90	MOVE BL, [0355]
	0075	0090	Not Used	0050	NV(0)	NZ(0)	PL(0)	0F68	0109	75	90	SUB AL, BL
	00E5	0090	Not Used	0050	OV(1)	NZ(0)	NG(1)	0F68	010B	75	90	JGE 0111
	00E5	0090	Not Used	0050	OV(1)	NZ(0)	NG(1)	0F68	0111	75	90	MOV [0354], AL
	00E5	0090	Not Used	0050	OV(1)	NZ(0)	NG(1)	0F68	0114	75	90	INT 20

Figure 13. Tracing Chart with Two Runs, With One Going Into the Loop

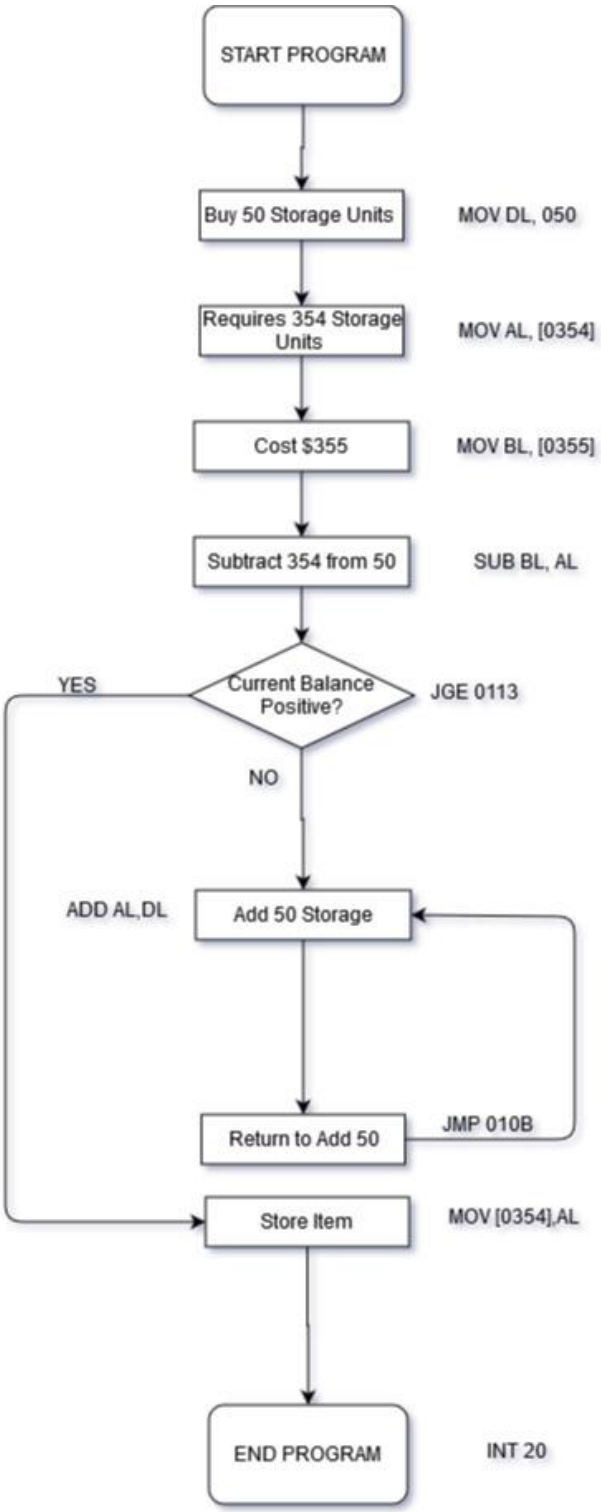

Figure 14. Flow with One JGE Instruction and Hand Assembly

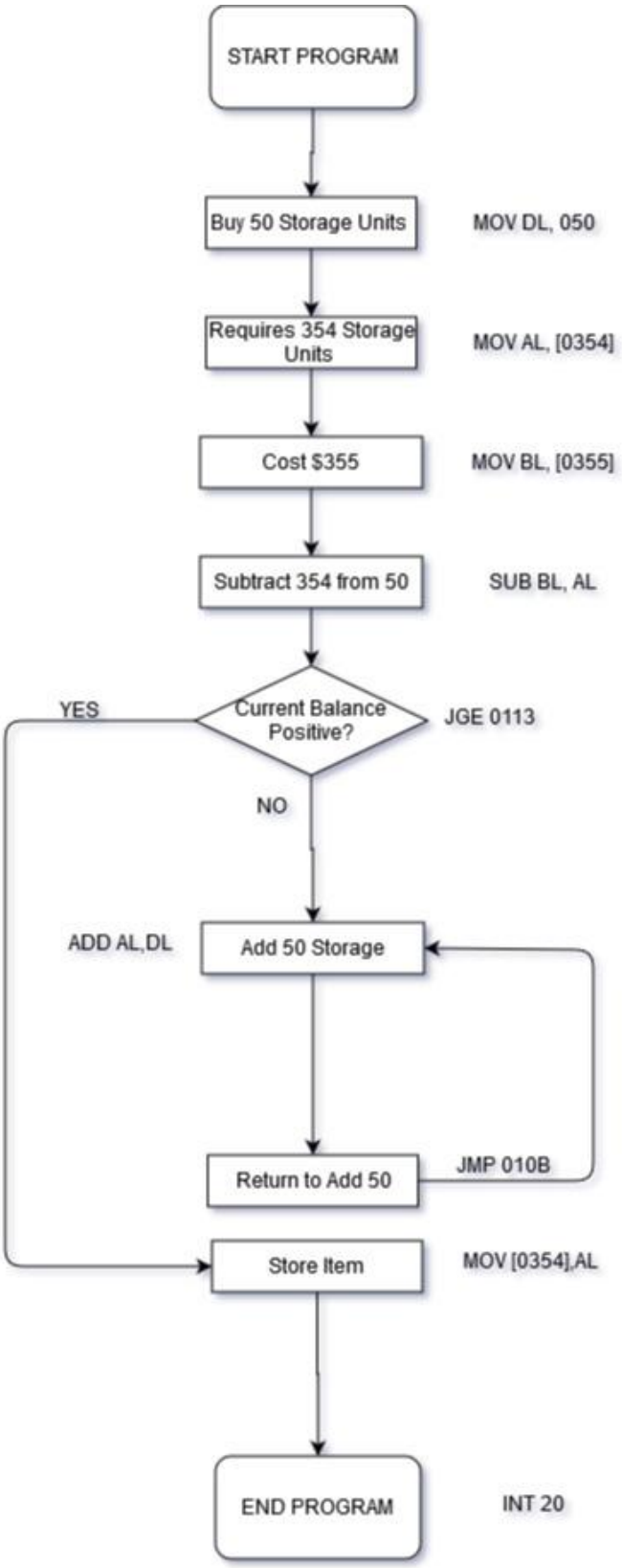

Figure 15. Flow Chart with Hand Assembly, Title, Counter, and One JGE Instruction

Instruction:	MOV DX, 0200								
Address:	CS	**100**		Operation:	MOV	Dest.:	DX	Source:	**200**
Instruction Format		1011 w reg: immediate data							
		w- 1	reg- 010						
Binary:	1101	1010	0002						
	B	A	2						
Hex:	BA0002								
Instruction:	MOV AH, 09								
Address:	CS	**103**		Operation:	MOV	Dest.:	AH	Source:	**9**
Instruction Format		1011 w reg: immediate data							
		w- 0	reg- 100						
Binary:	1101	0100	0009						
Hex:	B409								
Instruction:	INT 21								
Address:	CS	**105**		Operation:	INT	Dest.:	21	Source:	
Instruction Format		1100 1101: type							
		type- 21							
Binary:	1100	1101	0001 0101						
	C	D	21						
Hex:	CD21								
Instruction:	MOV DL, 050								
Address:	CS	**107**		Operation:	MOV	Dest.:	DL	Source:	**50**
		Immediate to reg							
Instruction Format		1011 w: reg immediate data							
		w- 0	reg- 010	immediate data- 50h					
Binary:	1011	0010	50h						
	B	2	50						
Hex:	B250								
Instruction:	MOV AL, [0354]								
Address:	CS	**109**		Operation:	MOV	Dest.	AL	Source:	**354**
		memory to AL							
Instruction Format		1010 000w: full displacement							
		w- 0							
Binary:	1010	0000	5403h						
	A	0	5403						
Hex:	A05403								
Instruction:	MOV BL, [0355]								
Address:	CS	**010C**		Operation:	MOV	Dest.	BL	Source:	**355**
Instruction Format		1000 101w: mod reg r/m							
		w- 0	mod- 00	reg- 011	r/m- 110				
Binary:	1000	1010	0001	1110	5503h				
	8	A	1	E	5503				
Hex:	8A1E5503								
Instruction:	SUB AL, BL								
Address:	CS	**110**		Operation:	SUB	Dest.:	AL	Source:	**BL**
Instruction Format		0010 100w: 11 reg1 reg2							
		w-0	reg 1- 011	reg2- 000					
Binary:	0010	1000	1101	1000					
	2	8	D	8					
Hex:	28D8								
Instruction:	INC by 1								
Address:	CS	**112**		Operation:	INC	Dest.	358	Source:	
Instruction Format		1111 111w: mod 000 r/m							
		w- 0	mod- 00	r/m- 110					
Binary:	1111	1110	0100	0110					
	F	E	0	6	5803				
Hex:	FE065803								

Instruction:	JGE 011C								
Address:	CS	**116**		Operation:	JGE	Dest.	011C	Source:	
Instruction Format		0111 tttn: 8-bit displacement							
		tttn- 1101							
Binary:	0111	1101							
	7	D	06h						
Hex:	7D04t								
Instruction:	ADD AL, DL								
Address:	CS	**118**		Operation:	ADD	Dest.	AL	Source:	**DL**
Instruction Format		0000 000w: 11 reg1 reg2							
		w- 0	reg1- 010	reg2-000					
Binary:	0000	0000	1101	0000					
	0	0	D	0					
Hex:	00D0								
Instruction:	JMP 0116								
Address:	CS	**011A**		Operation:	JMP	Dest.	116	Source:	
Instruction Format		1110 1011: 8-bit displacement							
Binary:	1110	1011	1111 1010						
	E	B	FA						
Hex:	EBFA								
Instruction:	MOV [0354], AL								
Address:	CS	**011C**		Operation:	MOV	Dest.	354	Source:	**AL**
		mem to AL							
Instruction Format		1010 001w: full displacement							
		w- 0							
Binary:	1010	0010	5403h						
	A	2	5403						
Hex:	A25403								
Instruction:	MOV DX, 0358								
Address:	CS	**011F**		Operation:	MOV	Dest.	DX	Source:	**358**
Instruction Format		1011 w reg: immediate data							
		w- 1	reg- 010						
Binary:	1011	1010	5803						
	B	A	5803						
Hex:	BA5803								
Instruction:	MOV AH, 09								
Address:	CS	**122**		Operation:	MOV	Dest.	9	Source:	**AH**
Instruction Format		1011 w reg: immediate data							
		w- 1	reg- 010						
Binary:	1011	1010	0009						
	B	A	9						
Hex:	BA09								
Instruction:	INT 21								
Address:	CS	**124**		Operation:	INT	Dest.	21	Source:	
		INT n							
Instruction Format		1100 1101: type							
		type- 21							
Binary:	1100	1101	0010 1001						
	C	D	21						
Hex:	CD21								
Instruction:	INT 20								
Address:	CS	**126**		Operation:	INT	Dest.	20	Source:	
		INT n							
Instruction Format		1100 1101: type							
		type- 20							
Binary:	1100	1101	0010 1000						
	C	D	20						
Hex:	CD20								

Figure 16. Hand Chart with Modified Program

```
#include <stdio.h>
#include <stdlib.h>

int main (void)
{
        int dl = 50;
        int al,bl,counter;
        printf("Hello my name is Mario, Welcome to CpE 185.\n\n");
        printf("Enter a value for AL register.\n");
        scanf ("%d", &al);
        printf("Enter a value for BL register.\n");
        scanf ("%d", &bl);
        al = bl - al;
        while(al <= 0)
        {
      printf ("\nThe value for AL register is: %i", al);
                counter = counter + 1;
                al = al + dl;
        }
        printf ("\n\nThe final value of AL register is: %i\n", al);
        printf ("The counter is equal to: %i", counter);
        return  EXIT_SUCCESS;
}
```

Figure 17. C Program with Title and Counter

Conclusion: After completing this part I gained a better understanding about how to do hand assembly. Seeing how each instruction has their own binary value assigned to it allowing it to be executed and later turn into hexadecimal.

Part3. Microsoft's Assembly Language Development System (MASM)

Description: Introduce the use of the assembler, while gaining experience with assembler syntax, Programmer's Workbench (PWB), and Code Viewer (Debugger).

Engineering Data: In the beginning we were given a code that prints Hello World ten times, as seen in the flowchart in figure 18. Then we were tasked to include a title and our last name, and it would also print 10 times, the flowchart can be seen in figure 19. Lastly we had to modify that code in order to take a user input, the flowchart can be seen in figure 20.

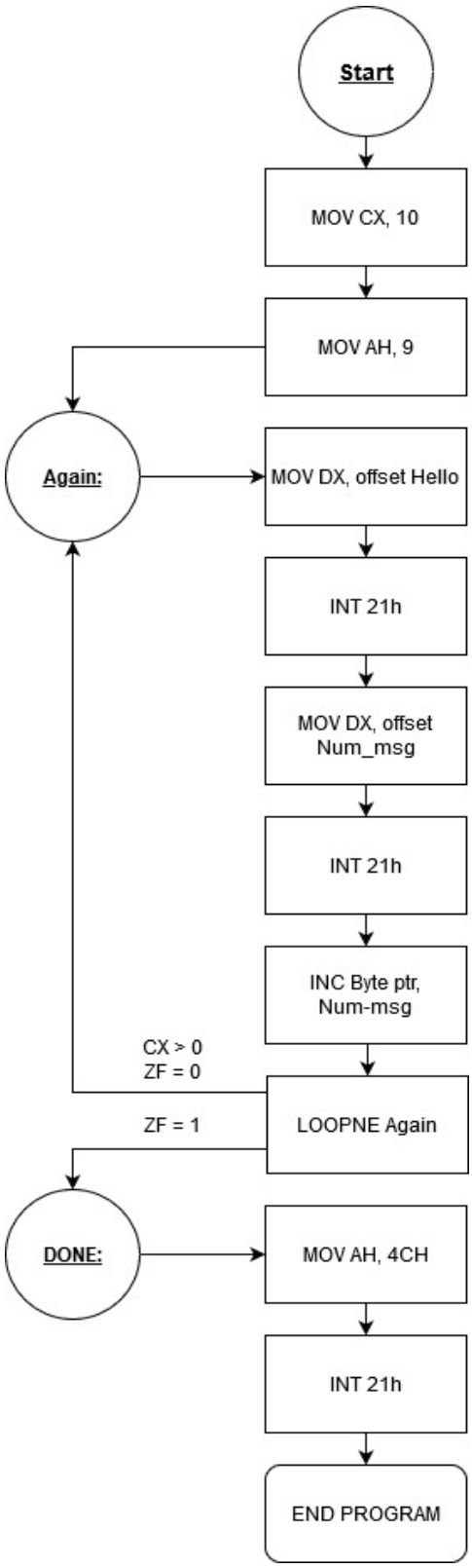

Figure 18. Flowchart for Original Code in NASM

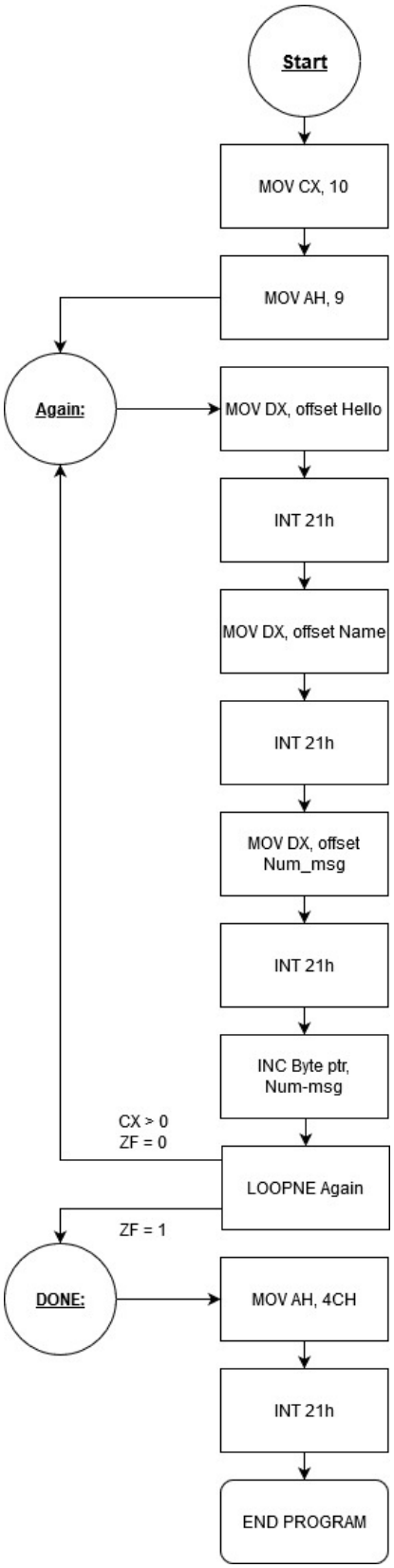

Figure 19. Flowchart of NASM Code with Last Name

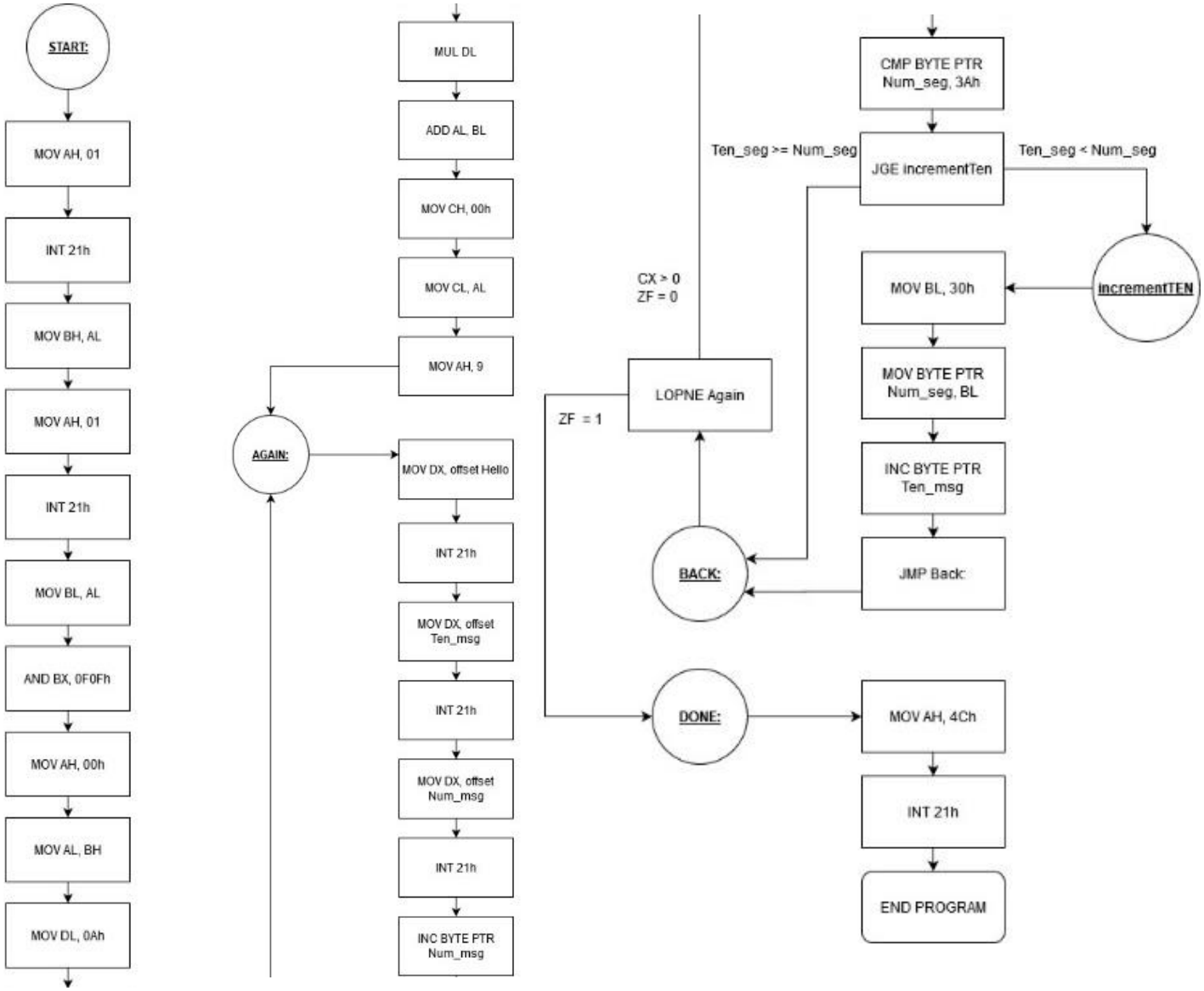

Figure 20. Flowchart of NASM Code with Last Name and User Input

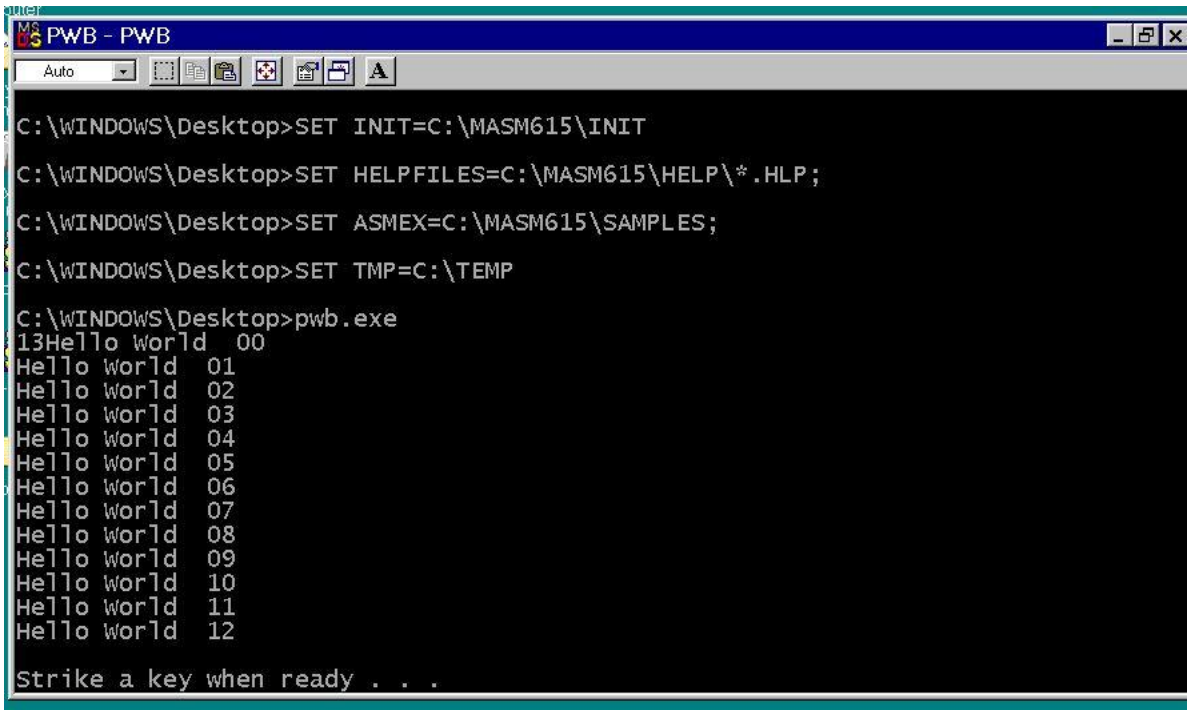

Figure 21. NASAM Terminal Output from Code

Once I ran the code it was evident that my code was running and could run “Hello World” for more than ten times, as seen in figure 21. After completing this the next step was to transfer our NASAM code into C language, while also inserting some inline assembly, the code can be seen in Figure 22.

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
        int count = 0;
        printf("Hello World Program with Title and User Input\n");
        printf("Mairo Palacios\n");
        printf("Please enter a number\n");
        scanf("%d", &count);
        for(int i = 0; i<count;)
        {
                __asm__ __volatile__ ("inc %%ecx;" : "+c" (i));
                printf("Hello World %d\n", i);
        }
        return 0;
}
```

Figure 22. C Programming with Inline Assembly

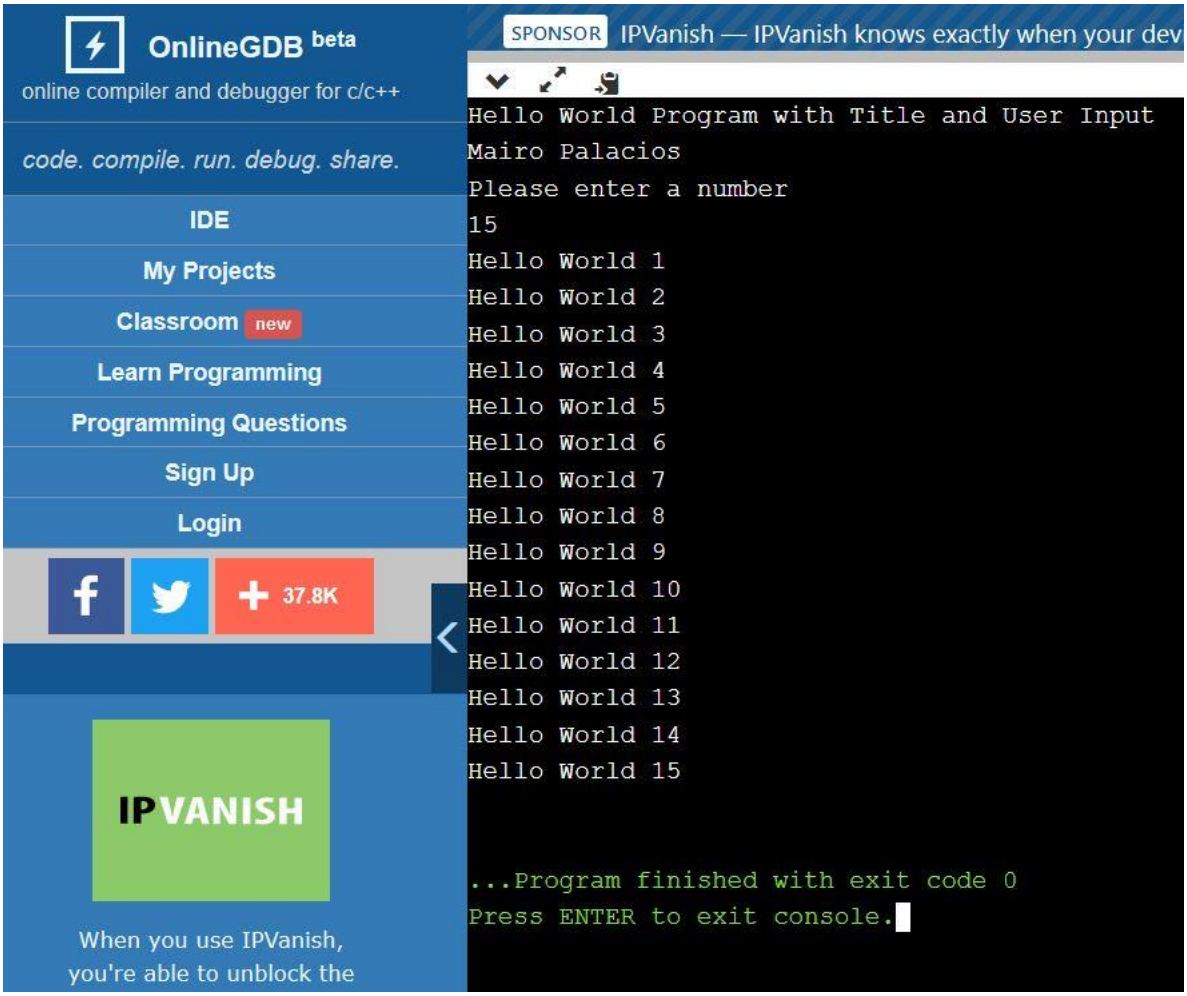

Figure 23. Output from C Program with Inline Assembly

Conclusion: At the end of this part of the lab I learned that PWB environment is not that different from DEBUG. When creating the NASAM code it was much easier because we did not have to turn our instructions to Hexadecimal in order to enter it into the DEBUG environment. As far as creating the C program, incrementing a value is a lot easier to do when you do not involve inline assembly.

Final Conclusion:

Overall this lab was a good exercise to review C programming more and do a more in-depth understanding about Assembly Language. This lab took me the longest not only to complete but to fully understand what each part is doing and how they are all connected. The hand assembly allowed us to visually see what a computer does when an instruction is being executed. The DEBUG environment allowed us to practice writing assembly but with the flow charts it made it easier to understand the logic behind assembly. Lastly the connection with C programming made us put what we learned into a higher level language, displaying how the assembly program works.