Shammah Thao

EEE 174 - CpE 185 Lab Section #2

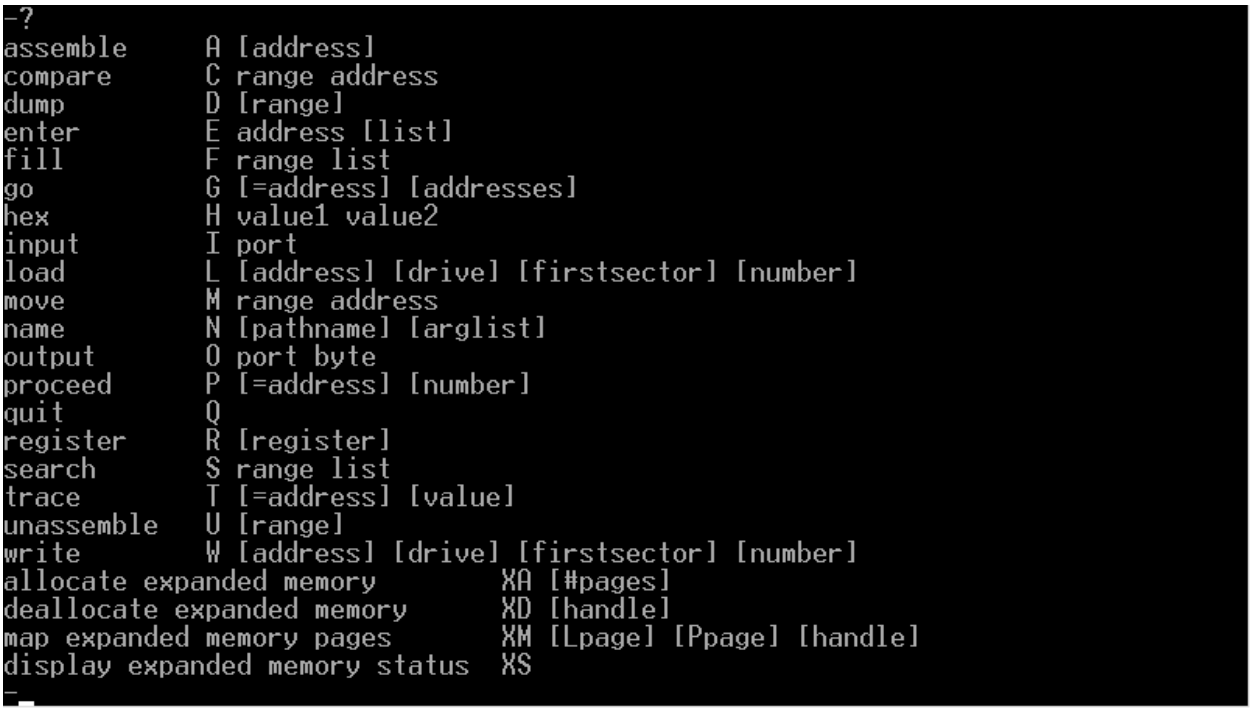
Monday & Wednesday

Lab 1: x86 and C refresher

Dahlquist

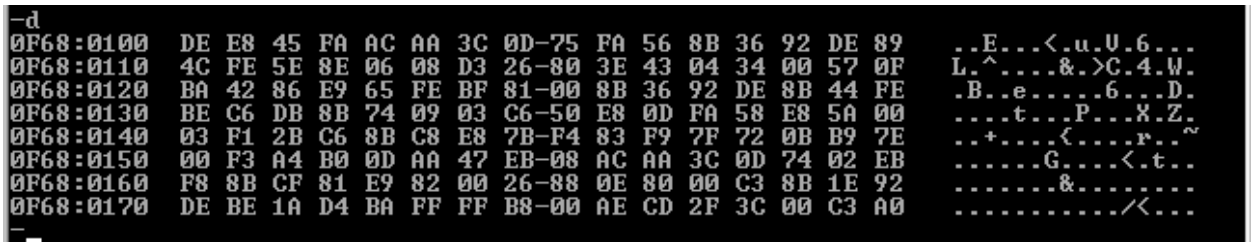
Part 1: Intro to debug and C refresher

1.

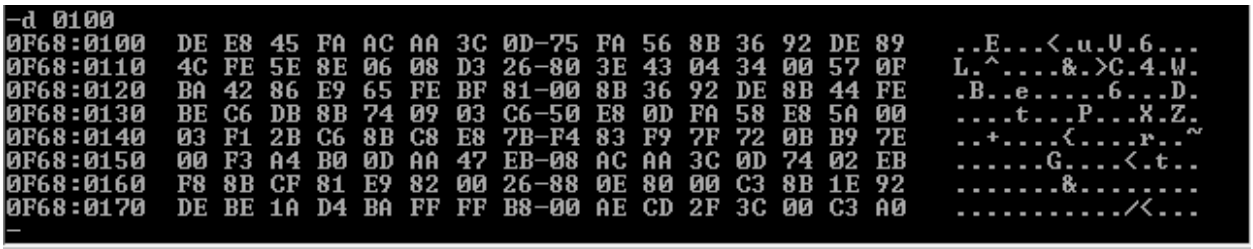


First thing we had to do was get access to a debugging program in a provided virtual machine. Going into the program called MS-Dos prompt we type “debug” then “?” to open a listing of commands.

2.



Using the command “d” (dump) we display content of the memory location.



Using d 0100 we focus on displaying listing from the memory location 0100 and so on.

```
-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
-
```

We then used d 0100 0110 to be even more specific on what memory location to view.

The two listing of number are basically like a between bars. Tells the program that you want to view the memory location between these two number

```
-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                                                .
-
```

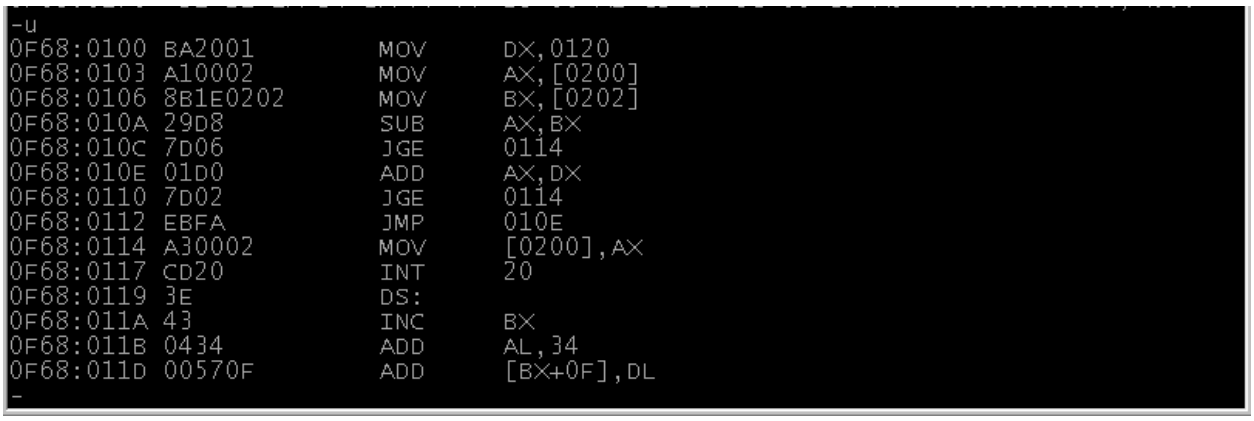
Following up, we used d 0100 0200, to display the number memory between 0100 and 0200.

3.

```
0F68:0112  5E.eb
-e0113
0F68:0113  8E.fa
-e0114
0F68:0114  06.a3
-e0115
0F68:0115  08.00
-e0115
0F68:0115  00.00
-e0116
0F68:0116  D3.02
-e0117
0F68:0117  26.cd
-e0118
0F68:0118  80.20
-d 0100
0F68:0100  BA 20 01 A1 00 02 8B 1E-02 02 29 D8 7D 06 01 D0   . ........).}...
0F68:0110  7D 02 EB FA A3 00 02 CD-20 3E 43 04 34 00 57 0F   }....... >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   ............./<...
-
```

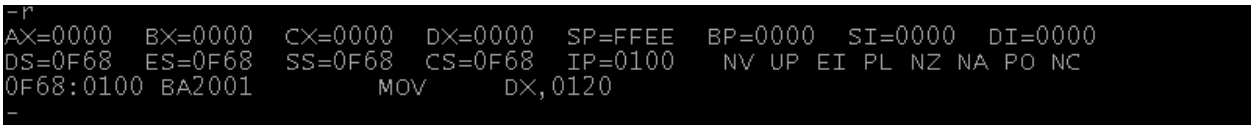
When then used the command “e” (enter) which allowed us to edit the memory location. we used e100 as the starting point then replaced it with the number we were given.

4.



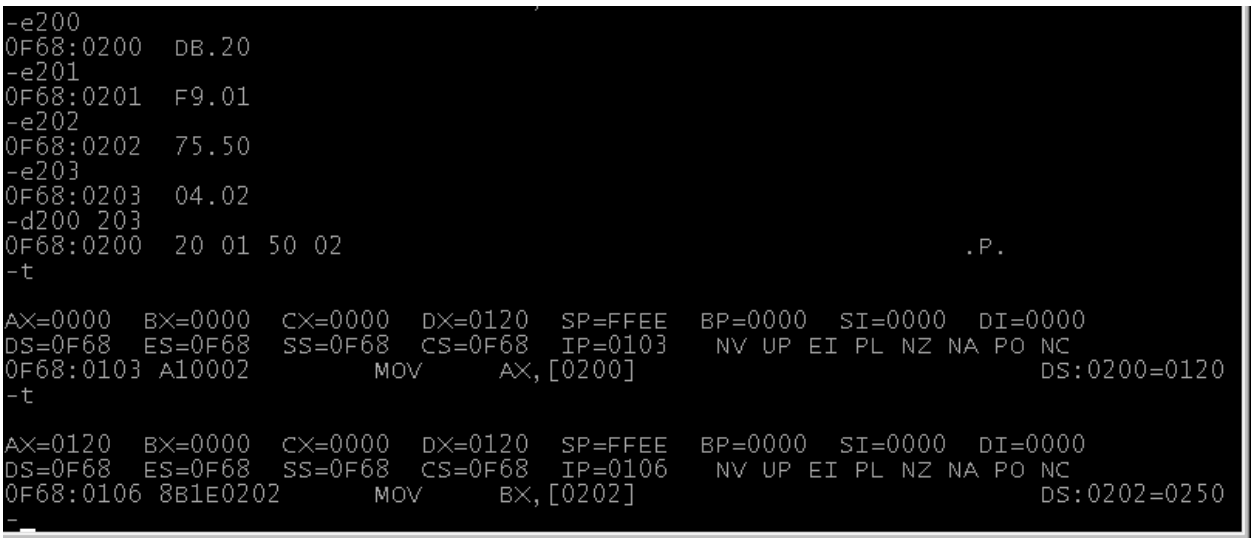
We then used the command “u” (unassembled) to view the changes that was done to the memory location. its tells us what was move to where and where the number were stored.

5.



Using “r” (register modify) we set a instruction pointer(IP) at the beginning of the program that we first edited which is 0100.

6.



we then used the “t” (trace), which basically shows that was done to the program set step by step.

```
-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
-e200
0F68:0200  10.82   00.22   75.72   22.22
-d200 203
0F68:0200  82 22 72 22                                          ."r"
-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=2282
-t

AX=2282  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=2272
-t

AX=2282  BX=2272  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_
```

This is one of the trace for the non loop one

```
AX=2282  BX=2272  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=0010  BX=2272  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
-t

AX=0010  BX=2272  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 NC
0F68:0114 A30002        MOV     [0200],AX                       DS:0200=2282
-t

AX=0010  BX=2272  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 NC
0F68:0117 CD20          INT     20
-
```

Until the ending at int 20

7.

```
-g

Program terminated normally
```

Using “g” (go), we were able to terminate out of the debugger. We also used “g” to break out specific breakpoint. Breaking at different addresses that we put in.

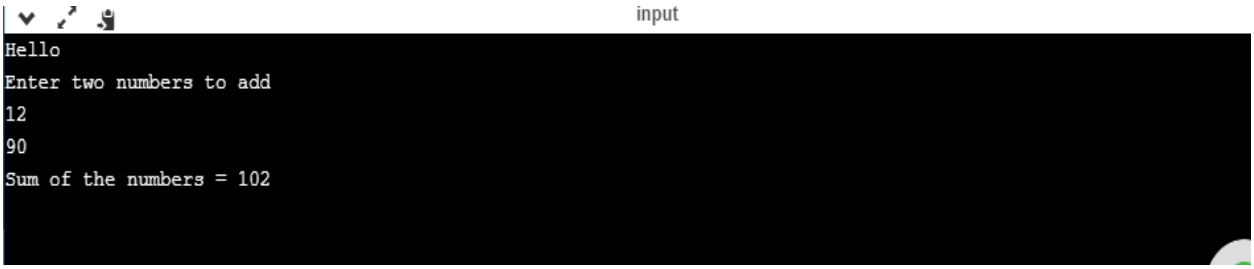
8. The function of the program is to basically tell you what the program is going to do, either its trying to move a memory location, or its trying to add or subtraction a memory value. Until you get to int 20h, which is the end program

C Refresher

Using www.onlinegdb.com , an online compiler for c,

```c
#include <stdio.h>

int main()
{
    //directly print out the word hello
    printf("Hello");

    //create variables for the number
    int x, y, z;

    //ask question for user input
    printf("\nEnter two numbers to add\n");
    scanf("%d%d", &x, &y);  // get user input and put it into x and y

    z = x + y; //add number

    printf("Sum of the numbers = %d\n", z); //print added number




    return 0;
}
```

Wrote hello in C program. Also wrong a code to take in user input to add two numbers.



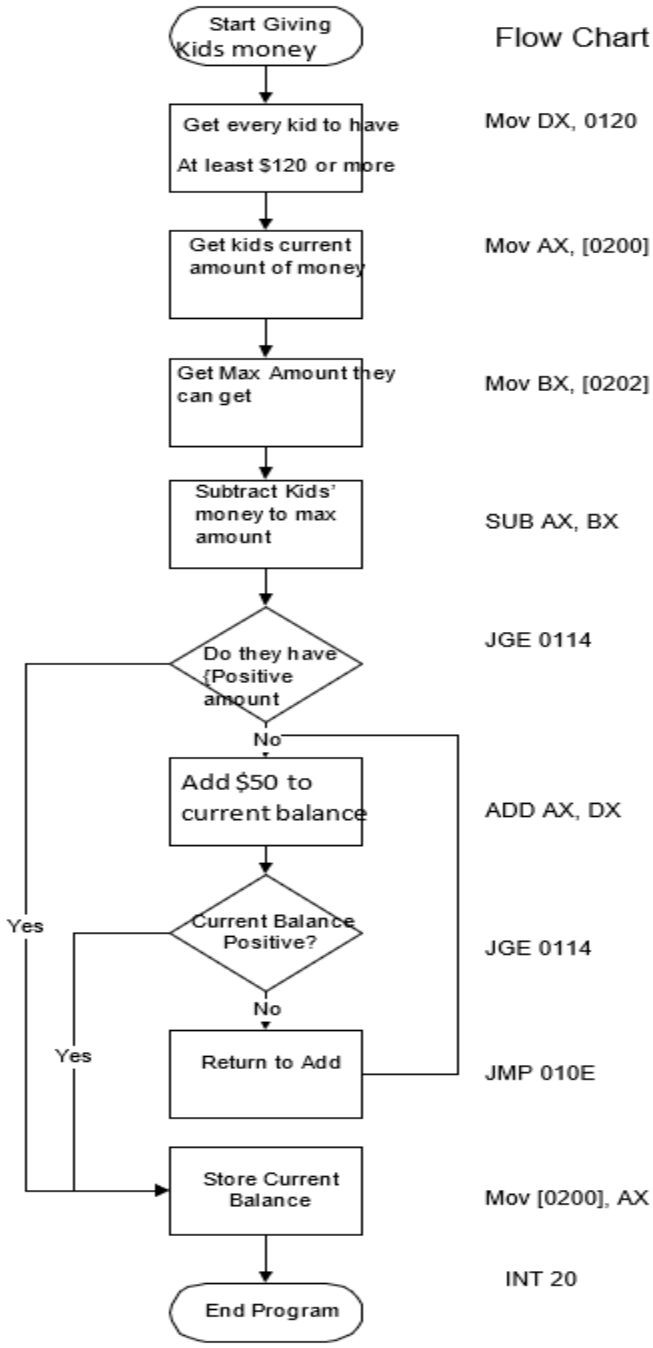
The output shows Hello and two number being entered and added together.

```c
#include <stdio.h>


int main() {

    //Registers
    int ax = 0,bx=0,dx=0;


    //Memory location at m200 = 333 & m202 = 555
    int m200 = 222;
    int m202 = 555;

    // now we will mov DX, 0120, meaning assgining dx to 333
    dx = m200;

    //MOV AX, [0200] - AX is assigned to content of memory location at m200
    ax = m200;

    //MOV BX, [0202] - BX is assigned to content of memory location at m202
    bx = m202;

    //SUB AX, BX - the subtraction of AX and BX is stored at AX
    ax = ax - bx;

    //JGE 0114 - Checking if AX is greater than 0; skips the while loop if true
    while (ax < 0){  // while ax is not greater than 0

        ax = ax + dx;// ADD AX, DX - the addition of AX and DX is stored at AX
        printf("%d\n", ax); //Print value of ax to track its changes.
    }
    //JMP 010E - If ax is still not greater than 0,jump back to the while loop

    m200 = ax;  // MOV [0200], AX - its value will be assigned to the content m200 when AX is greater than 0

    printf("%d\n", m200);
    //memory location at 200
    return 0; //INT 20 - terminate the program
}
```

For this code, I rewrote what was happening in the debugger in c language. With comments telling what is happening during each of the process.

```
-111
111
111


...Program finished with exit code 0
Press ENTER to exit console.
```

the result from the out shows the it had to go through the loop twice to double check that it is greater than 0. Also showing the final result of m200 at the end.



Using this flow chart to understand how to the debug work and its functionality with the codes that was input into it

CPE 185
Laboratory Hand-Assembly Template
Dahlquist/Stoffers/Schultz

Instruction:	MOV DX	120							
Address:	CS	**: 100**		Operation:	MOV	Dest.:	DX	Source:	**120**
Instruction Format		1011 wreg immediate data							
Binary:	1011	1010	2001h						
	B	A	2001						
Hex:	BA2001								
Instruction:	MOV	AX 0200							
Address:	CS	: 103		Operation:	MOV	Dest.:	AX	Source:	**200**
Instruction Format		1010 001w full displacement							
		w=1							
Binary:	1010	0001	0002						
	A	1	2						
Hex:	A10002								
Instruction:	MOV	BX	**202**						
Address:	CS	**: 106**		Operation:	MOV	Dest.:	BX	Source:	**202**
Instruction Format		memory to reg							
		1000 101w mod reg r/m							
		w=1	mod=00	reg=011	r/m=110				
Binary:	1000	1011	0001	1110	0202h				
Hex:	8B1E0202								
Instruction:	SUB	AX	BX						
Address:	CS	**: 10A**		Operation:	Sub	Dest.:	AX	Source:	**Bx**
Instruction Format									
		w=1	reg 1= 011	reg 2=000					
Binary:	0010	1001	1101	1000					
Hex:	29D8								
Instruction:	Jge 0114								
		Conditional	Jump						
Address:	CS	**: 10C**		Operation:	JGE	Dest.:	114	Source:	

Instruction Format		011 ttn 8-bit displacement							
		ttn=1101							
Binary:	0111	1101							
	7	D	D6h						
Hex:	7D06								
Instruction:	ADD	AX	**DX**						
Address:	CS	**:10E**		Operation:	Add	Dest.:	Ax	Source:	**Dx**
Instruction Format		0000 00w reg1 reg2							
		w=1	reg1=011	reg2=000					
Binary:	0000	0001	1101	0000					
Hex:	01D0								
Instruction	: Jge 0114								
Address:	CS	:0110			Operation:	Jge	Dest:	114	Source
Instruction	Format	011 ttn	8-bit	displacement					
		ttn=1101							
Binary	111	1101	10						
	7	D	2						
Hex	7D02								
Instruction:	Jmp	010E							
Address:	CS	**:0112**		Operation:	jmp	Dest.:	010E	Source:	
Instruction Format		1110 1011: 8 bit displacement							
		w=1	reg1=011	reg2=000					
Binary:	1110	1011	1111	1010					
Hex:	EBFA								
Instruction:	MOV	200	**Ax**						
Address:	CS	**:0114**		Operation:	Mov	Dest.:	200	Source:	**Ax**
Instruction Format		1000 100w mod reg r/m							
		w=1	reg= 011	r/m=110	mod=00				
Binary:	1000	1011	0011	0110					
	1000	0	11	110					
Hex:	A30002								
Instruction:	INT 20								

The following tracing chart was used then after first getting the instruction of the debugger. We would then try to get the hex which is the machine language that we would need.

Program Tracing Chart												
			Registers:									
RUN 1												
	AX:	BX:	CX:	DX:	OF:	ZF:	SF:	CS:	IP:	DS:200	DS:202	Next Instruction:
Value:--->	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	250	0	120	NV (0)	NZ (0)	PL (0)	0F68	010A	0120	250	SUB AX,BX
	FED0	250	0	120	NV (0)	NZ (0)	NG (1)	0F68	010C	0120	250	JGE 0114
	FED0	250	0	120	NV (0)	NZ (0)	NG (1)	0F68	010E	120	250	ADD AX, DX
	FED0	250	0	120	NV (0)	NZ (0)	NG (1)	0F68	110	120	250	JGE 0114
	FFF0	250	0	120	NV (0)	NZ (0)	NG (1)	0F68	112	120	250	JMP 010E
	FFF0	250	0	120	NV (0)	NZ (0)	NG (1)	0F68	010E	120	250	ADD AX, DX
	FFF0	250	0	120	NV (0)	NZ (0)	NG (1)	0F68	110	120	250	JGE 0114
	110	250	0	120	NV (0)	NZ (0)	PL (0)	0F68	114	120	250	MOV [0200],AX
	110	250	0	120	NV (0)	NZ (0)	PL (0)	0F68	117	110	250	INT 20
						0						
						0						
RUN 2												
	AX:	BX:	CX:	DX:	OF:	ZF:	SF:	CS:	IP:	DS:200	DS:202	Next Instruction:
Value:--->	0000	0000	0000	0000	NV (0)	NZ (0)	PL (0)	0F68	0100	2282	2272	Dx,0120
	0000	0000	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	0103	2282	2272	Mov Ax[0200]
	2282	0000	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	0106	2282	2272	Mov Bx[0202]
	2282	2272	0000	120	NV (0)	NZ (0)	PL (0)	0F68	010A	2282	2272	sub Ax,Bx
	10	2272	0000	120	NV (0)	NZ (0)	NG (1)	0F68	010C	2282	2272	JGE 0114
	10	2272	0000	120	NV (0)	NZ (0)	NG (1)	0F68	010E	2282	2272	MOV [0200],AX
	10	2272	0000	120	NV (0)	NZ (0)	NG (1)	0F68	110	2282	2272	INT 20

As i went the program, we had to use tracing in the debugger to look step by step on what changed in our register and flags. In the tracing chart, one did a loop and the other just checked if the number was big enough and just skipped through the loop after the first initial check.

Part2: Hand assembly and C programming

To start, we had to set up a hand assembly, with modified register. We are suppose to use our own unqie register, mine was 354. Using 354, we replaced the original from part 1.

CPE 185

Laboratory Hand-Assembly Template

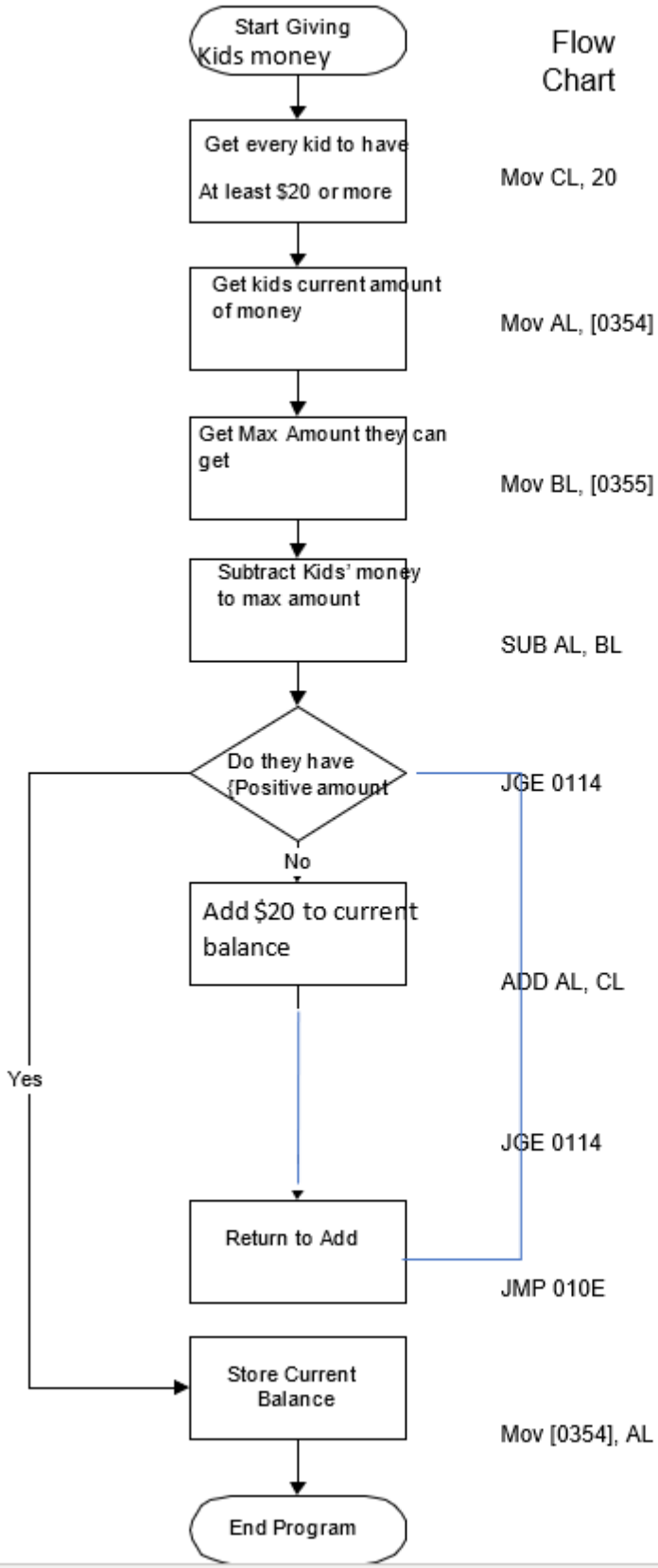
Dahlquist/Stoffers/Schultz

Instruction	MOV CL	120							
Address:	CS	**: 100**		Operation:	MOV	Dest.:	CL	Source:	**120**
Instruction Format		1011 wreg immediate data							
Binary:	1011	1010	2001h						
	B	A	2001						
Hex:	B120								
Instruction	MOV	AL 0200							
Address:	CS	: **103**		Operation:	MOV	Dest.:	AL	Source:	**354**
Instruction Format		1010 001w full displacement							
		w=0							
Binary:	1010	0001	0002						
	A	1	2						
Hex:	A05403								
Instruction	MOV	BL	**202**						
Address:	CS	**: 106**		Operation:	MOV	Dest.:	BL	Source:	**355**
		memory to reg							
Instruction Format		1000 101w mod reg r/m							
		w=0	mod=00	reg=011	r/m=110				
Binary:	1000	1011	0001	1110	0202h				
Hex:	8A1E5503								
Instruction	SUB	AL	BL						
Address:	CS	**: 10A**		Operation:	Sub	Dest.:	AL	Source:	**BL**
Instruction Format									
		w=0	reg 1= 011	reg 2=000					
Binary:	0010	1001	1101	1000					

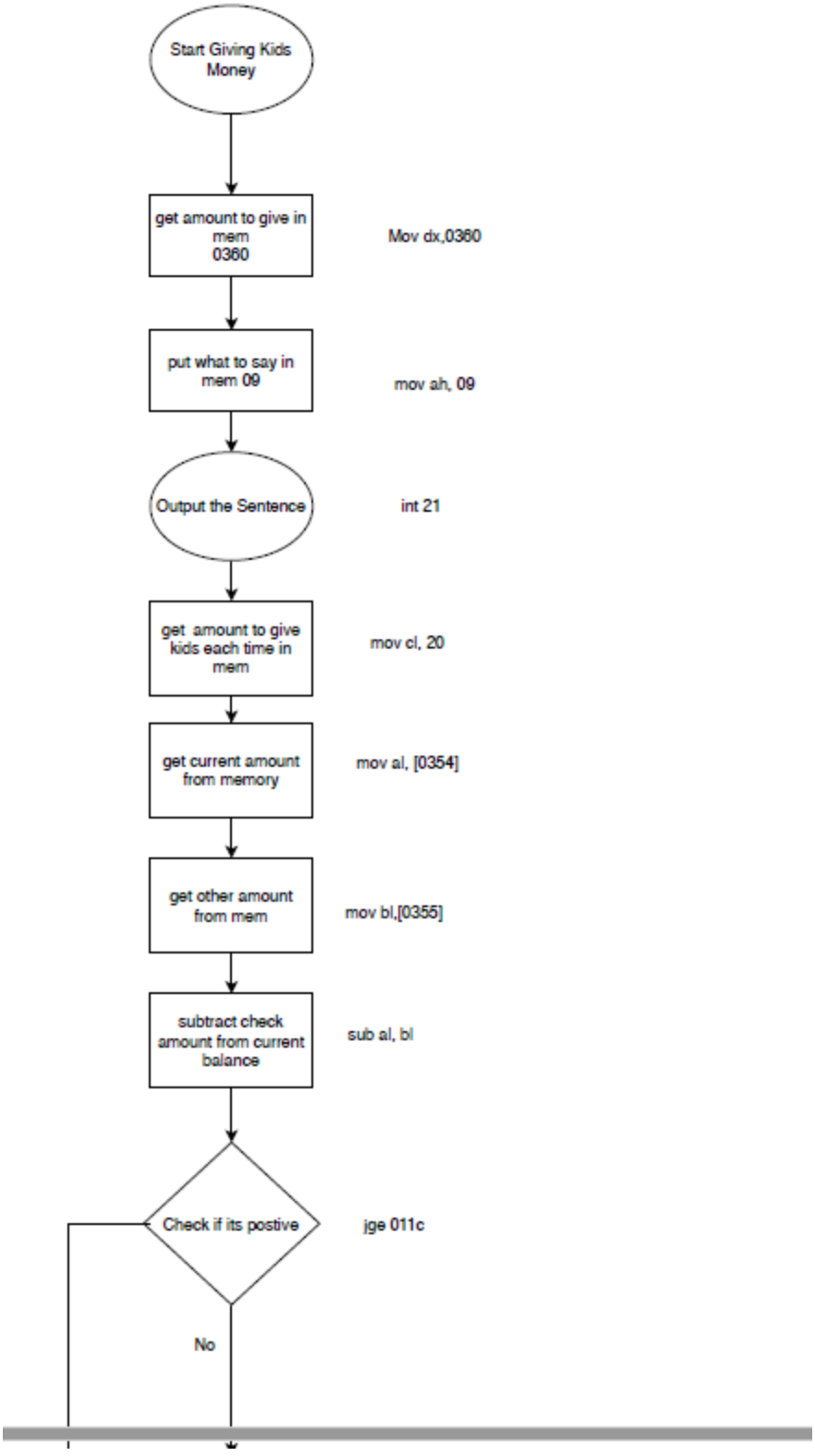
Hex:	28D8								
Instruction	Jge 0114								
		Conditional	Jump						
Address:	CS	**: 10C**		Operation:	JGE	Dest.:	114	Source:	
Instruction Format		011 ttn 8-bit displacement							
		ttn=1101							
Binary:	0111	1101							
	7	D	D6h						
Hex:	7D04								
Instruction	ADD	AL	**CL**						
Address:	CS	**:10E**		Operation:	Add	Dest.:	AL	Source:	**CL**
Instruction Format		0000 00w reg1 reg2							
		w=0	reg1=011	reg2=000					
Binary:	0000	0001	1101	0000					
Hex:	00C8								
Instruction	Jmp	010E							
Address:	CS	**:0112**		Operation:	jmp	Dest.:	010E	Source:	
Instruction Format		1110 1011: 8 bit displacement							
		w=0	reg1=011	reg2=000					
Binary:	1110	1011	1111	1010					
Hex:	EBFA								
Instruction	MOV	200	**AL**						
Address:	CS	**:0114**		Operation:	Mov	Dest.:	354	Source:	**AL**
Instruction Format		1000 100w mod reg r/m							
		w=0	reg= 011	r/m=110	mod=00				
Binary:	1000	1011	0011	0110					
	1000	0	11	110					
Hex:	A25403								

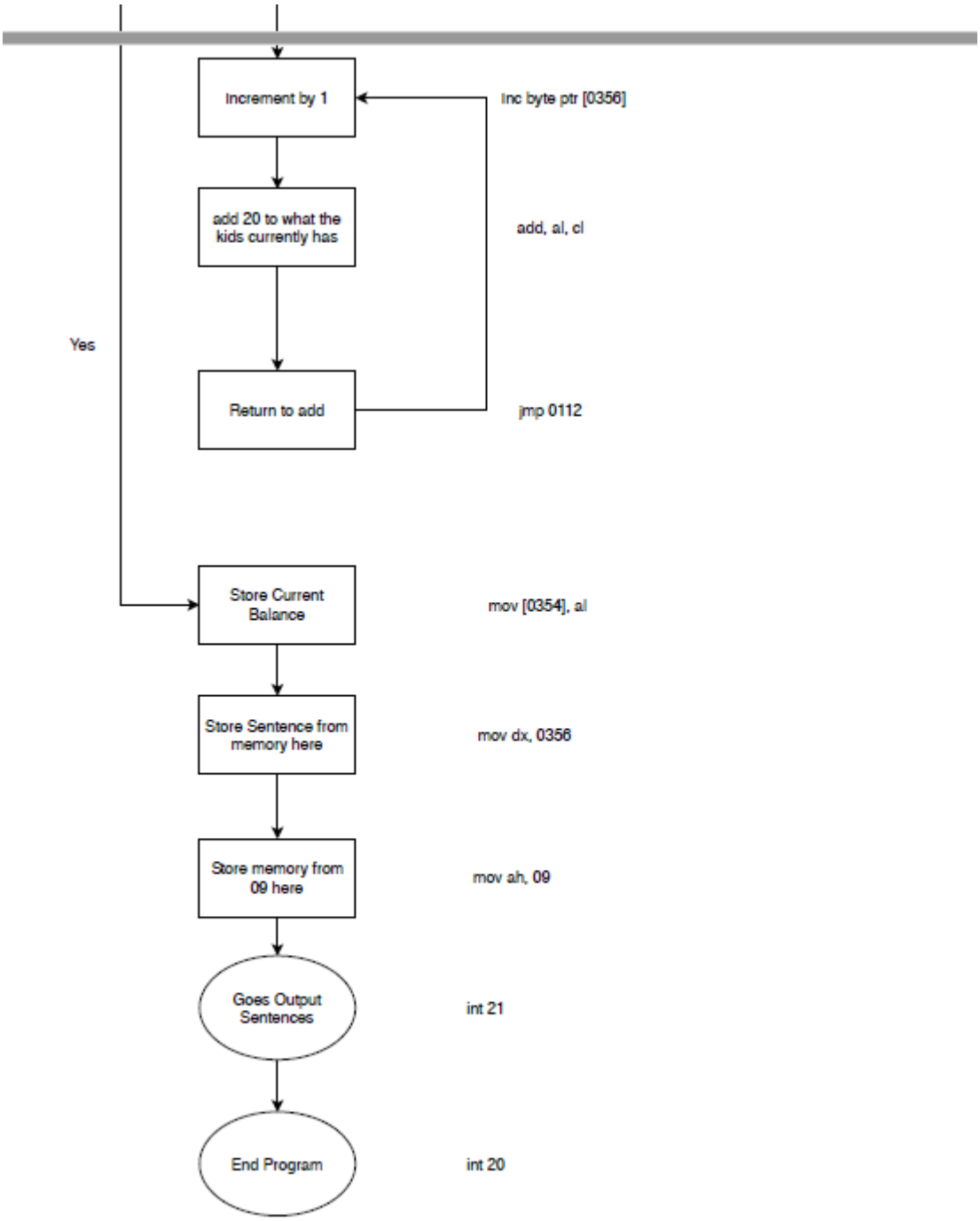
Hex:	A25403								
Instruction	INT 20								
Address:	CS	: 0117			Operation:	INT	Dest.:	20	Source:
		INT n – Interrupt Type n							
Instruction Format		1100 1101 : type							
		n=20 or 0010 0000							
Binary:	1100	1101	0010	0000					
Hex:	CD 20								

The flow chart below, was made with the same intent as the one from part 1 of this lab. Just modifying the register to match the code. Following the sequence of how the code should be executed.


Start Giving Kids money
Flow Chart
Get every kid to have At least $20 or more
Mov CL, 20
Get kids current amount of money
Mov AL, [0354]
Get Max Amount they can get
Mov BL, [0355]
Subtract Kids' money to max amount
SUB AL, BL
Do they have {Positive amount
JGE 0114
No
Add $20 to current balance
ADD AL, CL
Yes
JGE 0114
Return to Add
JMP 010E
Store Current Balance
Mov [0354], AL
End Program
INT 20

The second flow care is a flow chart that included the increment and creating a title for the debugger.


Start Giving Kids Money
get amount to give in mem 0360
Mov dx,0360
put what to say in mem 09
mov ah, 09
Output the Sentence
int 21
get amount to give kids each time in mem
mov cl, 20
get current amount from memory
mov al, [0354]
get other amount from mem
mov bl,[0355]
subtract check amount from current balance
sub al, bl
Check if its postive
jge 011c
No



The tracing chart below was used to trace the program after the title and increment was included into the program. We let it ran and over time the sign flag would change then revert back while not being in the loop

Program Tracing Chart												
			Registers:									
RUN 1					offset	zero flag						
	AL:	BL:	CL:	DX:	OF:	ZF:	SF:	CS:	IP:	DS:354	DS:355	Next Instruction:
Value:--->	0924	0009	0020	0360	NV (0)	NZ (0)	PL (0)	0F68	0107	5	0009	mov cl, 20
	0924	0009	0020	0360	NV (0)	NZ (0)	PL (0)	0F68	0109	5	0009	mov al,[0354]
	0905	0009	0020	0360	NV (0)	NZ (0)	PL (0)	0F68	10C	5	0009	mov bl,[0355]
	0905	9	0020	0360	NV (0)	NZ (0)	PL (0)	0F68	110	5	0009	sub al, bl
	09FC	9	0020	0360	NV (0)	NZ (0)	NG (1)	0F68	112	5	0009	jge 011C
	09FC	9	0020	0360	NV (0)	NZ (0)	NG (1)	0F68	114	5	0009	inc byte ptr [0356]
	09FC	9	0020	0360	NV (0)	NZ (0)	NG (1)	0F68	118	5	0009	add al, cl
	091C	9	0020	0360	NV (0)	NZ (0)	NG (1)	0F68	11A	5	0009	jmp 0112
	091C	9	0020	0360	NV (0)	NZ (0)	NG (1)	0F68	112	5	0009	mov [0354], al
	091C	9	0020	0360	NV (0)	NZ (0)	NG (1)	0F68	11C	5	0009	mov dx, 0356
	091C	9	0020	0356	NV (0)	NZ (0)	PL (0)	0F68	11F	5	0009	mov ah,09
	091C	9	0020	0356	NV (0)	NZ (0)	PL (0)	0F68	117	5	0009	int 21
						0						
						0						
RUN 2												
	AL:	BL:	CL:	DX:	OF:	ZF:	SF:	CS:	IP:	DS:354	DS:355	Next Instruction:
Value:--->	0924	0000	0000	0360	NV (0)	NZ (0)	PL (0)	0F68	0107	13	0009	mov cl, 20
	0924	0000	0020	0360	NV (0)	NZ (0)	PL (0)	0F68	0109	13	0009	mov al,[0354]
	091C	0000	0020	0360	NV (0)	NZ (0)	PL (0)	0F68	10C	13	0009	mov bl,[0355]
	091C	9	0020	0360	NV (0)	NZ (0)	PL (0)	0F68	110	13	0009	sub al, bl
	913	9	0020	0360	NV (0)	NZ (0)	PL (0)	0F68	112	13	0009	jge 011C
	913	9	0020	0360	NV (0)	NZ (0)	PL (0)	0F68	011C	13	0009	mov [0354], al
	913	9	0020	0360	NV (0)	NZ (0)	PL (0)	0F68	11F	13	0009	mov dx, 0356
	913	9	0020	0356	NV (0)	NZ (0)	PL (0)	0F68	122	13	0009	mov ah,09
	913	9	0020	0356	NV (0)	NZ (0)	PL (0)	0F68	124	13	0009	int 21

The hand assemebly was made to include the loop and the incrementing to get the machine code needed for the demo.

									Shammah	Thao	Unqiue	354
	EEE 174											
	Laboratory Hand-Assembly Template											
	Dahlquist/Stoffers/Schultz											
	Instruction	mov dx,	354									
	Address:	CS	: **0100**		Operation	mov		Dest.:	dx	Source:	**354**	
			immediate to register									
	Instruction Format		1011 w reg immediate data									
			w = 0	reg=010	data=1101010100							
	Binary:	1011	0010	1101010100								
		B	2	354								
	Hex:	B2 35 40										
	Instruction	mov ah	9									
	Address:	CS	**105**		Operation	mov		Dest.:	ah	Source:	**9**	
			immediate to register(alternate en									
	Instruction Format		1011 w reg immediate data									
		1011	w=0	reg= 100	data=9							
	Binary:	1011	0100	1001								
		B49	4	9								
	Hex:	B49										
	Instruction	mov	cl	**20**								
	Address:	CS	: **107**		Operation	mov		Dest.:	cl	Source:	**20**	
			immediate to reg(alternate encodi									
	Instruction Format		1011 w reg immediate data									
		1011	w=0	reg=001	data=100000							
	Binary:	1011	0001	0010	0000							
		B	1	2	0							
	Hex:	B120										
	Instruction	mov	al	**[0354]**								
	Address:	CS	: **0109**		Operation	mov		Dest.:	al	Source:	**[0354]**	
			memory to register									
	Instruction Format		0000 001w mod reg r/m									
			w=0	mod=00	reg=000	r/m=110						
	Binary:	0000	0010	0000	0000	0110						
		0	2	0	0	6						
	Hex:	A05403					2006					
	Instruction	mov	bl	**[0355]**								
	Address:	CS	: **010C**		Operation	mov		Dest.:	bl	Source:	**[0355]**	
			memory to reg									
	Instruction Format		0000 001w mod reg r/m									
											1.0001E+31	
	Binary:	0000	0010	0000	0000	1110						
		0	2	0	0	E						

	Binary:	0000	0010	0000	0000	1110				
		0	2	0	0	E				
	Hex:	8A1E5503					0200E			
	Instruction	sub	al	**bl**						
	Address:	CS	: **0110**		Operation	sub	Dest.:	al	Source:	**bl**
			reg2 to reg1							
	Instruction Format		1000 101w 11 reg1 reg2							
			w=0	reg1= al=000	reg2=bl=011					
	Binary:	1000	1010	1100	0011			10100011011000		
		8	A	C	3					
	Hex:	28D8					8AC3			
			4							
	Instruction	jge	10E							
	Address:	CS	: **0112**		Operation:		Dest.:		Source:	
			jump if condition is met		8 displacement					
	Instruction Format		0111 tttn 8 bit displacement							
		111	tttn=1101	displacement= 6						
	Binary:	0111	1101	0000 0110						
		7	D	0	6			1.1111E+14		
	Hex:	7D08					7D06			
	Instruction	inc	byte	**ptr**	[0356]					
	Address:	CS	: **0114**		Operation	inc	Dest.:	[0356]	Source:	
			inc memory							
	Instruction Format		1111 111w mod 000 r/m							
		w=0	mod= 00	0	r/m=110					
	Binary:	1111	1110	0000	0011	0000				
		F	E	0	3	0		1.11111E+31		
	Hex:	FE065603					FE030			
	Instruction	add	al	**cl**						
	Address:	CS	: **0118**		Operation	add	Dest.:	al	Source:	**cl**
			reg2 to reg1							
	Instruction Format		0000 001w 11 reg1 reg2							
			w=0	reg1= 000	reg2=001					
	Binary:	0000	0010	1100	0001			11001000		
		0	2	C	1					
	Hex:	00C8					02C1			
	Instruction	jmp	10E							
	Address:	CS	: **011A**		Operation	jmp	Dest.:	10E	Source:	
			JMP -Unconditional Jump (to same segment)							
	Instruction Format		short 1110 1011 : 8-bit displacement							
		111	tttn=							
	Binary:	1110	1011	1111	1010			1.1101E+15		
		E	B	F	A					
	Hex:	EBF6					EBFA			
	Instruction	mov	[0454]	**al**						
	Address:	CS	: **011C**		Operation	mov	Dest.:	[0354]	Source:	**al**
			reg to mem							

Address:	CS	: **011C**		Operation	mov	Dest.:	[0354]		Source:	**al**	
		reg to mem									
Instruction Format		1000 100w mod reg r/m									
		w=0	mod=00	reg=000	r/m=110						
Binary:	1000	1000	0000	0110							
							1.01E+23				
Hex:	A25403					8806					
Instructior	mov	dx	**[0356]**								
Address:	CS	: **011F**		Operation	mov	Dest.:	dx		Source:	**[0356]**	
			mem to reg								
Instruction Format		1000 101w mod reg r/m									
		w=0	mod=00	reg 000	r/m=110						
Binary:	1000	1010	0000	0110							
							1.0111E+23				
Hex:	BA5603					8A06					
Instructior	mov	ah	**9**								
Address:	CS	: **0122**		Operation	mov	Dest.:	ah		Source:	**9**	
		immediate to register(alternate encoding)									
Instruction Format		1011 w reg immediate data									
	1011	w=0	reg= 100	data=9							
Binary:	1011	0100	1001								
							1.01101E+15				
Hex:	B409				B49						
Instructior	int	21									
Address:	CS	: **0124**		Operation	INT	Dest.:		21	Source:		
		INT n - Interrupt Type n									
Instruction Format		1100 1101: type									
		n=21 or 100001									
Binary:	1100	1101	1000	0001							
Hex:	CD 21										
Instructior	int	20									
Address:	CS	: **0126**		Operation	INT	Dest.:		20	Source:		
		INT n - Interrupt Type n									
Instruction Format		1100 1101: type									
		n=20 or 0010 0000									
Binary:	1100	1101	0010	0000							
Hex:	CD 20										

Following through the steps first part of the lab instruction, we were able to make it excute with g=100 without any issue. The last output shows that the original number 5 have changed to 1C due to the reg going through the loop.

```
-e354
0F68:0354  8D.05   A1.09
-d0354 355
0F68:0350              05 09                                           ..
-u100 115
0F68:0100 B120          MOV     CL,20
0F68:0102 A05403        MOV     AL,[0354]
0F68:0105 8A1E5503      MOV     BL,[0355]
0F68:0109 28D8          SUB     AL,BL
0F68:010B 7D04          JGE     0111
0F68:010D 00C8          ADD     AL,CL
0F68:010F EBFA          JMP     010B
0F68:0111 A25403        MOV     [0354],AL
0F68:0114 CD20          INT     20
-d0354 355
0F68:0350              05 09                                           ..
-g=100

Program terminated normally
-d0354 355
0F68:0350              1C 09                                           ..
-
```

Next on the instruction, we had it so it would be about to output a title from the debugger. Inputting in the instruction and the quote into the registers, we were about to output a title for our code.

```
0F68:0112 jge 0118
0F68:0114 add al, cl
0F68:0116 jmp 0112
0F68:0118 mov [0354], al
0F68:011B int 20
0F68:011D
-e354
0F68:0354  1C.05   09.09
-e360
0F68:0360  00.00
-e360 "EEE 174/ CPE 185 Lab 1 Part 2" 0d 0a "$"
-d360
0F68:0360  45 45 45 20 31 37 34 2F-20 43 50 45 20 31 38 35   EEE 174/ CPE 185
0F68:0370  20 4C 61 62 20 31 20 50-61 72 74 20 32 0D 0A 24    Lab 1 Part 2..$
0F68:0380  C3 73 FD 9C 53 51 56 57-55 06 1E 50 52 B4 59 CD   .s..SQVWU..PR.Y.
0F68:0390  21 59 5B BA 54 82 3D 41-00 74 04 8B C3 8B D1 1F   !Y[.T.=A.t......
0F68:03A0  07 5D 5F 5E 59 5B 9D C3-E8 D6 FF CB 56 57 51 BF   .]_^Y[......VWQ.
0F68:03B0  33 DA 33 C9 8B C1 57 AC-3C 00 74 05 AA 41 AC EB   3.3...W.<.t..A..
0F68:03C0  F7 B0 0D AA 5F 0E 1F 89-0E E1 D7 89 3E E3 D7 59   ...._.......>..Y
0F68:03D0  5F 5E C3 56 BF E3 E3 8B-4F 05 89 0E DD E2 51 F3   _^.V....O.....Q.
-g=100
EEE 174/ CPE 185 Lab 1 Part 2

Program terminated normally
-
```

For the final part, we included the incrementing and title into the debugger. We were about to get it to increase. I used the number 1 and 55 in the register which made it looped 3x

```
0F68:0128
-e360 "EEE 174/ CPE 185 Lab 1 Part 2" 0d 0a "$"
-
-e356 30 0d 0a "$"
-
-e354
0F68:0354  2E.01   A1.55
-g=100
EEE 174/ CPE 185 Lab 1 Part 2
3

Program terminated normally
-
```

For the C coding that involves, putting in the incrementing and the title, we just had to make a variable for the string and create a variable to count the number of increment in the loop.

```c
#include <stdio.h>


int main() {

    //Registers
    int al = 0,bl=0,cl=0;
    char m360[] = ("EEE 174/ CPE 185 Lab 1 Part 2"); //define memory 360

    printf("%s\n", m360); // print the string from memory 360


    //Memory location at m200 = 222 & m202 = 555
    int m354 = 1;
    int m355 = 55;
    int m356 =0;

    // now we will mov cl, 20, meaning assgining cl to 20
    cl = 20;

    //MOV al, [0354] - al is assigned to content of memory location at m354
    al = m354;

    //MOV bl, [0355] - bl is assigned to content of memory location at m355
    bl = m355;

    //SUB al, bl - the subtraction of al and bl is stored at al
    al = al - bl;

    //JGE 0114 - Checking if al is greater than 0; skips the while loop if true
    while (al < 0){  // while al is not greater than 0

        al = al + cl;// ADD al, cl - the addition of al and cl is stored at al
        m356++;  //increments how many time it went through the loop
    }
    //JMP 010E - If al is still not greater than 0,jump back to the while loop

    m354 = al;  // MOV [0200], al - its value will be assigned to the content m200 when al is greater than 0
    printf("%d", m356); // int 21 - print out info from mem 0356
    //memory location at 200
    return 0; //INT 20 - terminate the program
}
```

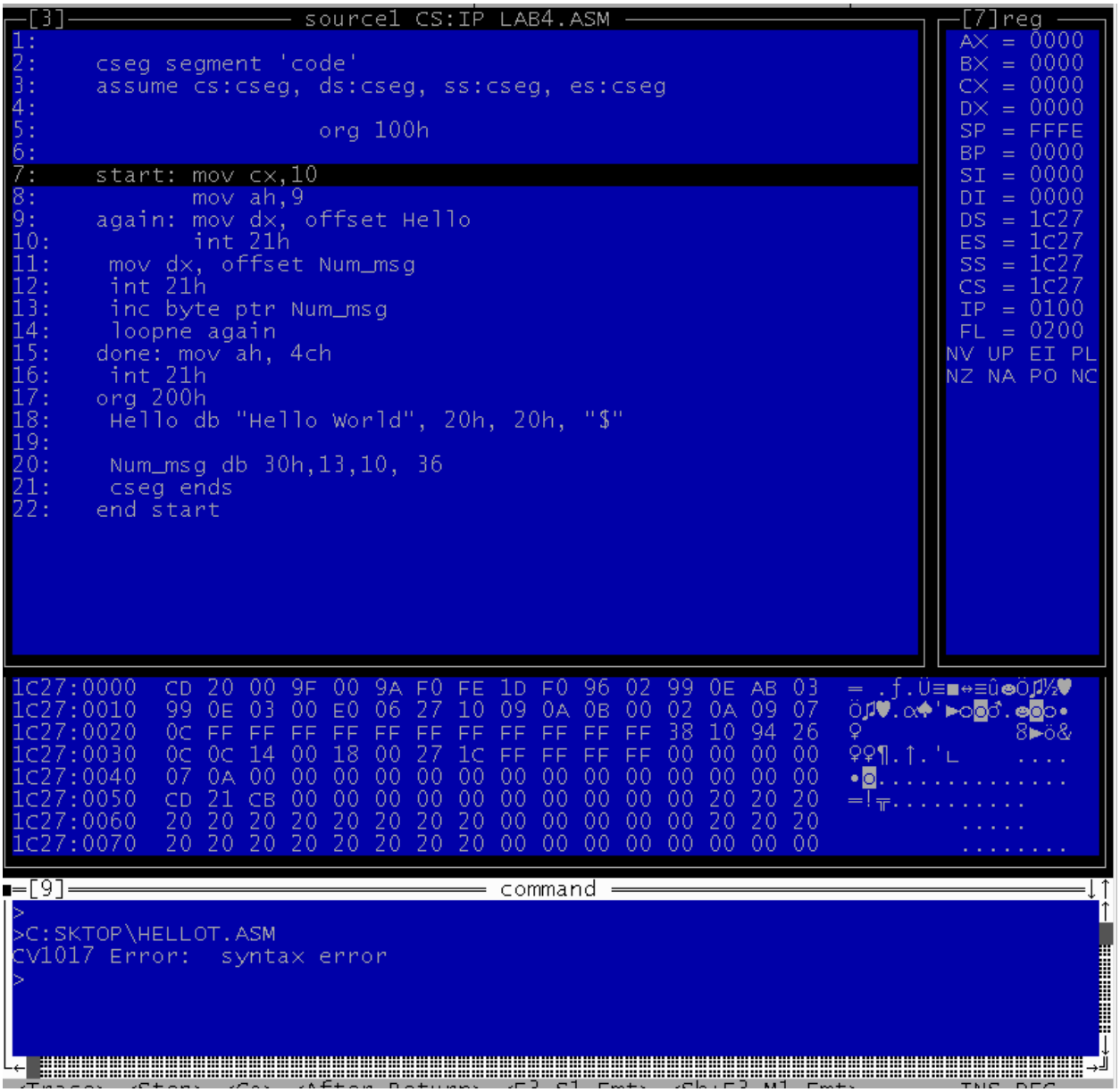
```
EEE 174/ CPE 185 Lab 1 Part 2
3

...Program finished with exit code 0
Press ENTER to exit console.
```

Part 3: Microsoft's Assembly language Development System (MASM)

For the beginning part of the lab, we about to set up the pwd's editor using the given instruction. We then set the given code into an asm file which ran the Following output.

```
C:\WINDOWS\Desktop>SET LIB=C:\MASM615\LIB;

C:\WINDOWS\Desktop>SET INCLUDE=C:\MASM615\INCLUDE;

C:\WINDOWS\Desktop>SET INIT=C:\MASM615\INIT

C:\WINDOWS\Desktop>SET HELPFILES=C:\MASM615\HELP\*.HLP;

C:\WINDOWS\Desktop>SET ASMEX=C:\MASM615\SAMPLES;

C:\WINDOWS\Desktop>SET TMP=C:\TEMP

C:\WINDOWS\Desktop>pwb.exe
Hello World  0
Hello World  1
Hello World  2
Hello World  3
Hello World  4
Hello World  5
Hello World  6
Hello World  7
Hello World  8
Hello World  9

Strike a key when ready . . .
```

Testing out the debugger, one of the feature of the pwd, we ever able to see step by step of what was happening during the code ran, by pressing F10 we were able to step through it.



One of the issue with the coding was that it couldn't be able to run twice, doing so will mess up the number. Reason being is that the number might be saved in the register, so when you output it the second time it would just increase from there.

To fix the issue we had to fix the coding so it would reset back to 0 after the first loop.

```
Program terminated normally
-u100
0F8B:0100 B90A00        MOV     CX,000A
0F8B:0103 B409          MOV     AH,09
0F8B:0105 BA0002        MOV     DX,0200
0F8B:0108 CD21          INT     21
0F8B:010A BA0E02        MOV     DX,020E
0F8B:010D CD21          INT     21
0F8B:010F FE060E02      INC     BYTE PTR [020E]
0F8B:0113 E0F0          LOOPNZ  0105
0F8B:0115 C6060E0230    MOV     BYTE PTR [020E],30
0F8B:011A B44C          MOV     AH,4C
0F8B:011C CD20          INT     20
0F8B:011E 0000          ADD     [BX+SI],AL
-
```

Doing so we were able to get the program to run without any issues

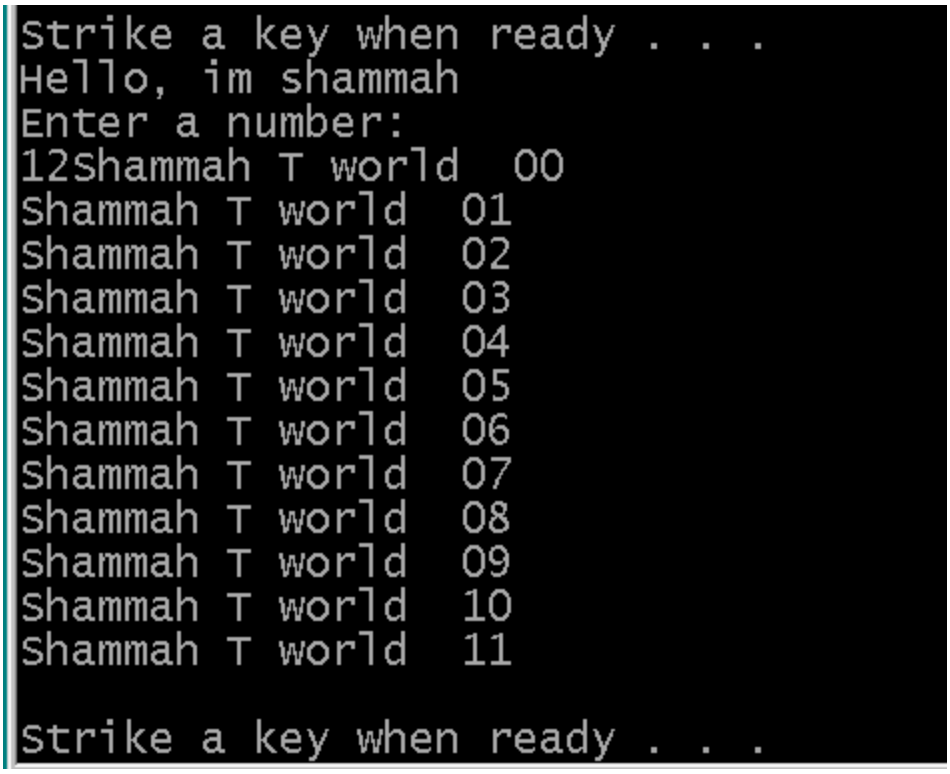
```
Hello World  1
Hello World  2
Hello World  3
Hello World  4
Hello World  5
Hello World  6
Hello World  7
Hello World  8
Hello World  9

Program terminated normally
-g=100
Hello World  0
Hello World  1
Hello World  2
Hello World  3
Hello World  4
Hello World  5
Hello World  6
Hello World  7
Hello World  8
Hello World  9

Program terminated normally
-
```

After fixing the issue we changed the quote to our name and see how it run with different character length

```
ShammahThao  1
ShammahThao  2
ShammahThao  3
ShammahThao  4
ShammahThao  5
ShammahThao  6
ShammahThao  7
ShammahThao  8
ShammahThao  9

Program terminated normally
-g=100
ShammahThao  0
ShammahThao  1
ShammahThao  2
ShammahThao  3
ShammahThao  4
ShammahThao  5
ShammahThao  6
ShammahThao  7
ShammahThao  8
ShammahThao  9

Program terminated normally
-
```

For the final part we had to make it so it takes in user input along with a title. We were able to make it so it output the correct number without much issue expect the issue with the input number and quote being on the same line.

```
start:
        mov ah,09h
        mov dx, offset prompt           ;get prompt
        int 21h                         ; output prompt
        mov ah,01h                      ; user input
        int 21h
        sub al, 30h                     ; subtract 30h form user input
        mov bl, 0ah
        mul bl                          ;multiple bl to ah, make into a 2 digit number
        mov bl,al
        mov ah, 01h
        int 21h                         ;get second digit accepted into asm
        sub al, 30h
        add bl, al                      ;add the first and second together
        mov cl, bl
        jmp again                       ; goes to again

again:
        mov ah,09h
        mov dx, offset Hello            ; get quote from hello
        int 21h
        mov dx, offset tens             ; get first digit
        int 21h
        mov dx, offset ones             ; get second digit
        int 21h
        inc byte ptr ones               ; increment by one
        mov al, ones                    ; move ones to al
        cmp al, 3ah                     ; compare 3a to al
        jg reset                        ; jump to reset if al is greater than al
        loopne again

reset:
        mov byte ptr ones, 30h          ;set 30h to ones
        inc byte ptr tens               ; increment tens by 1
        cmp cx,0                        ; compare reg to 0
        jg again                        ; jmp to again if its greater than 0
        jmp done                        ; jump to done otherwise

done:
        mov ah, 4ch                     ; end programn
        int 21h

org 200h

prompt db "Hello, im shammah", 0ah, 0dh, "Enter a number: ", 20h, 20h , 0ah, 0dh, "$"
hello db "Shammah T world", 20h, 20h, "$"
ones db 30h, 13, 10, "$"
tens db 30h, "$"

cseg ends
end start
```



At this part, we made a pure C coding of what was happening in the pwd code. I make it so mine accept 2 single integer and add them together to get a 2 digit num and put it inside a for loop to run.

```c
#include <stdio.h>

int main()
{
    int x, y,z;
    printf("Enter a single number : ");
    scanf("%d", &x);
    x = x * 10;
    printf("Enter the second Number: ");
    scanf("%d", &y);
    z = x + y;

 //   printf("Enter a 2 digi number: ");
 //   scanf("%d", &x);

    for(int i=0; i < z; i++)
    {
        printf("\nShammah T World %d", i);
    }
}
```

input

```
Enter a single number : 1
Enter the second Number: 2

Shammah T World 0
Shammah T World 1
Shammah T World 2
Shammah T World 3
Shammah T World 4
Shammah T World 5
Shammah T World 6
Shammah T World 7
Shammah T World 8
Shammah T World 9
Shammah T World 10
Shammah T World 11

...Program finished with exit code 0
Press ENTER to exit console.
```

For this part of the coding, we had to put inline assembly inside C. So to do this we have to use __asm__ along with some assembly code. I made it so mine teared apart a 2 digit number then

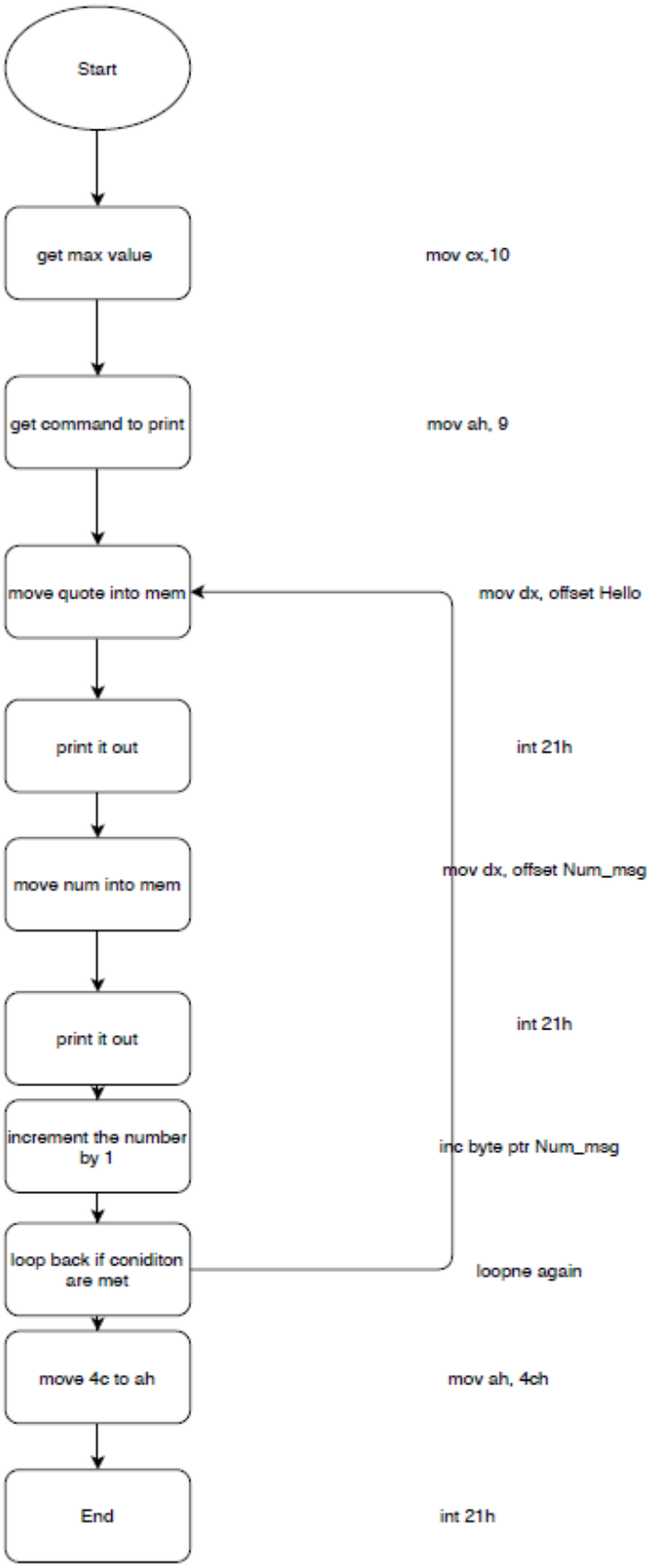
put in the asm which would multiply 10 to make it a 10 digit number, then add the second digit and then it was put inside a loop to print out the same output as the pwd.

```c
#include <stdio.h>

int main() {
    int x,y,z,add;
    printf("Enter a 2 digi number: ");
    scanf("%d", &x);
    y = x/10;
    printf("%d\n", y);
    z = x%10;
    printf("%d\n", z);

    __asm__ ( "imull %%ebx, %%eax;" : "=a" (y) : "a" (y) , "b" (10) );
    __asm__ ( "addl %%ebx, %%eax;" : "=a" (add) : "a" (y) , "b" (z) );

    printf( "%d + %d = %d\n", y, z, add );
    for(int i=0; i < add;i++)
    {
       printf("\nShammah T World %d", i);
    }
    return 0 ;
}
```

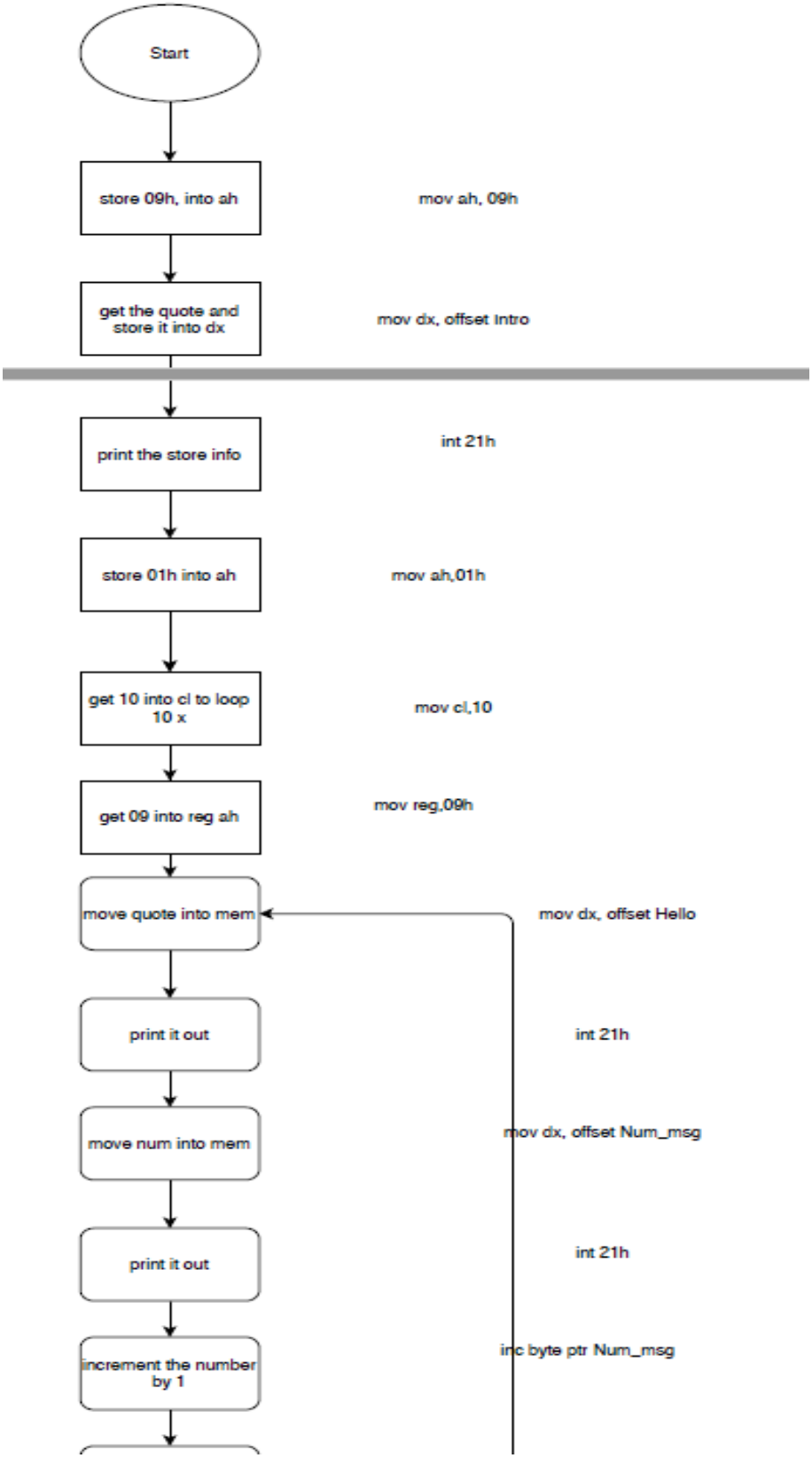
input

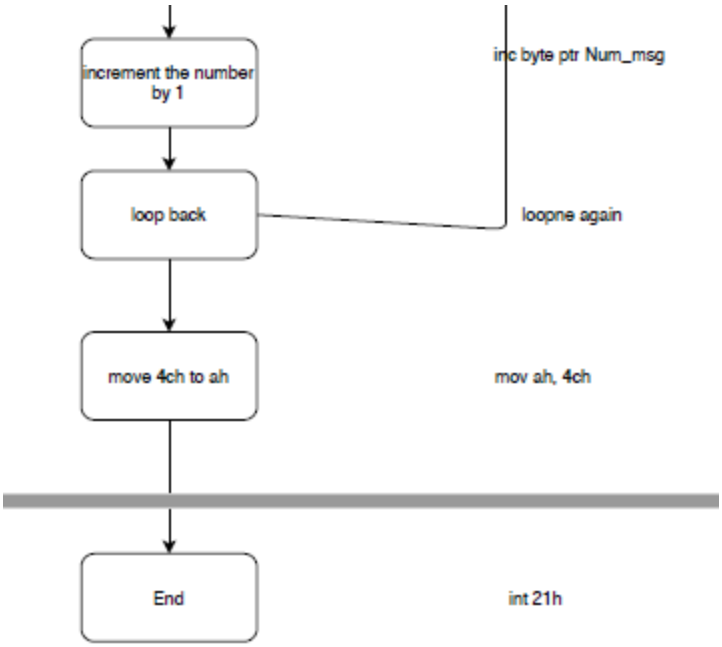
```
Enter a 2 digi number: 12
1
2
10 + 2 = 12

Shammah T World 0
Shammah T World 1
Shammah T World 2
Shammah T World 3
Shammah T World 4
Shammah T World 5
Shammah T World 6
Shammah T World 7
Shammah T World 8
Shammah T World 9
Shammah T World 10
Shammah T World 11

...Program finished with exit code 0
Press ENTER to exit console.
```

For the Flowchart of this lab, we have to include 3 flow chart. One with just the original code method, second is including the title in it, and lastly including the user input and incrementing the number.

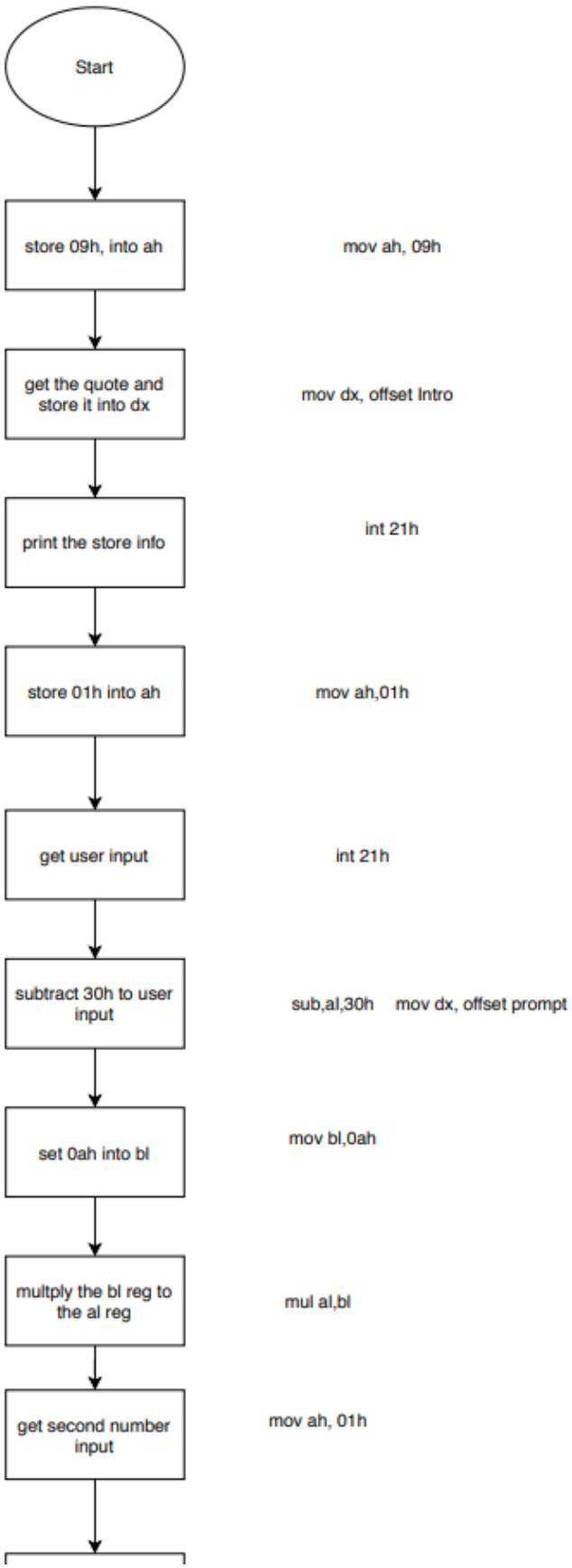

Start
get max value
mov cx,10
get command to print
mov ah, 9
move quote into mem
mov dx, offset Hello
print it out
int 21h
move num into mem
mov dx, offset Num_msg
print it out
int 21h
increment the number by 1
inc byte ptr Num_msg
loop back if coniditon are met
loopne again
move 4c to ah
mov ah, 4ch
End
int 21h

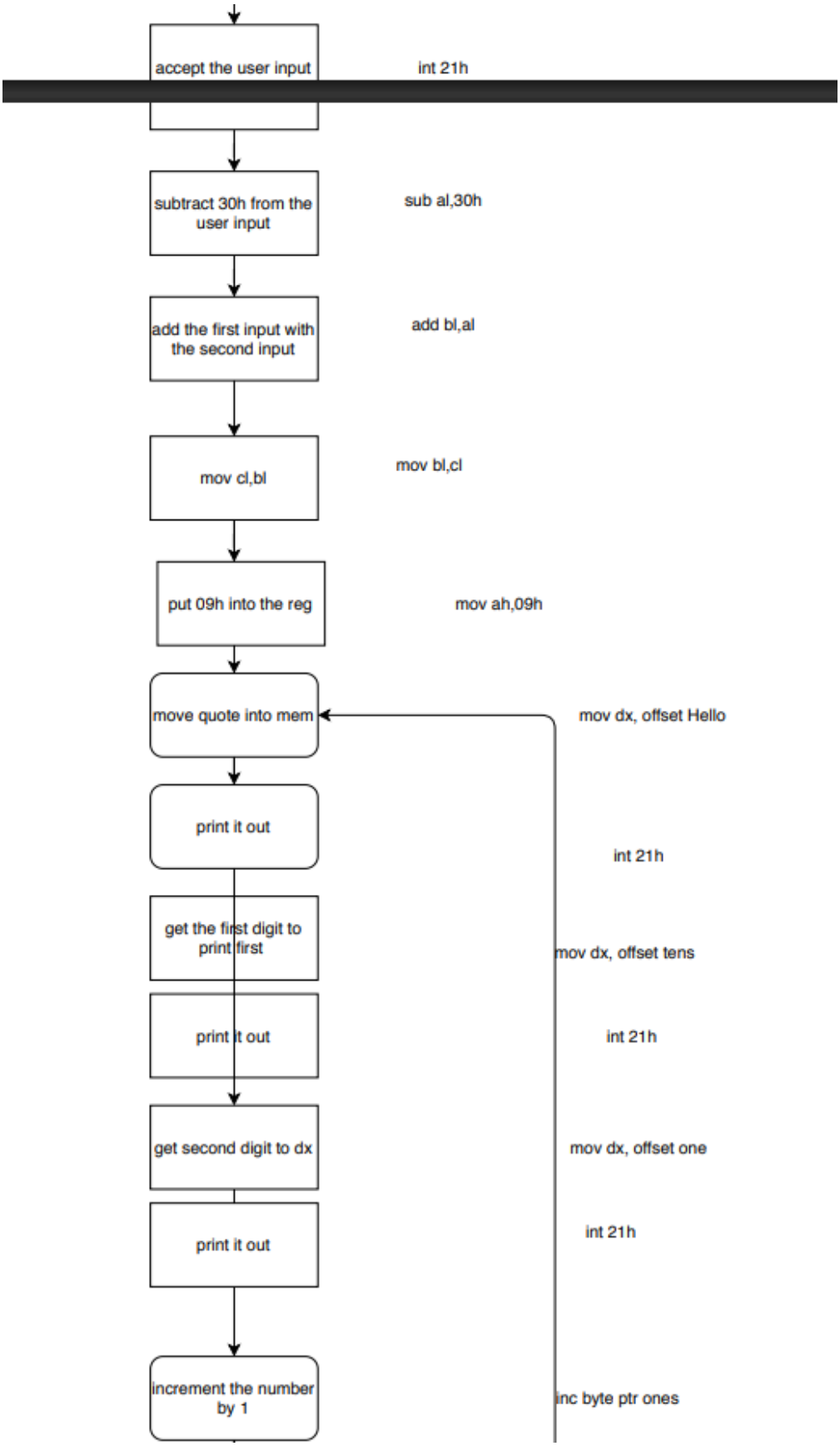
The flowchart above is the original method of how this pwd program would run

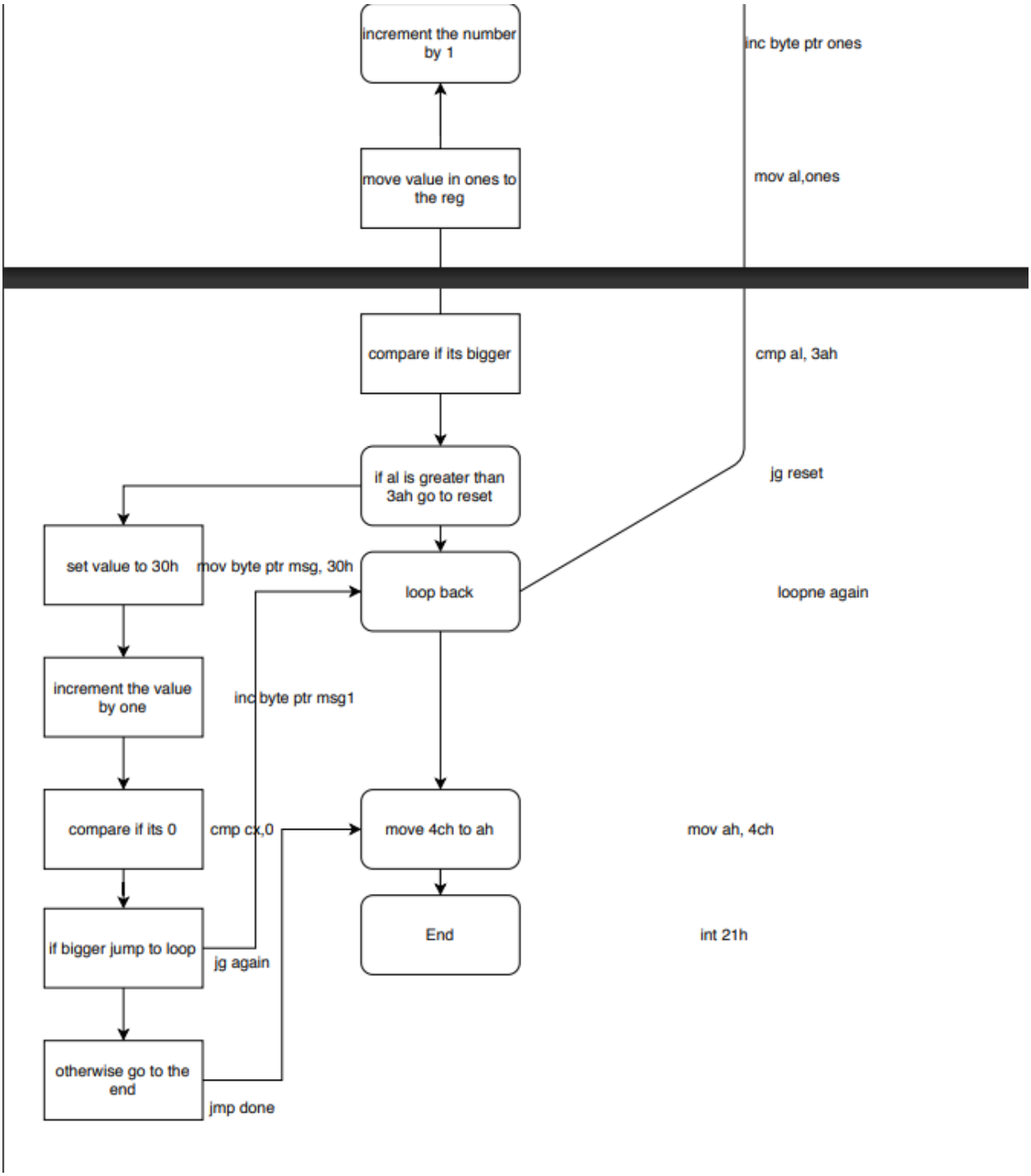

Start
store 09h, into ah
mov ah, 09h
get the quote and store it into dx
mov dx, offset Intro
print the store info
int 21h
store 01h into ah
mov ah,01h
get 10 into cl to loop 10 x
mov cl,10
get 09 into reg ah
mov reg,09h
move quote into mem
mov dx, offset Hello
print it out
int 21h
move num into mem
mov dx, offset Num_msg
print it out
int 21h
increment the number by 1
inc byte ptr Num_msg



The Second flow chart, above this is the method that included the incrementing and the title.


Start
store 09h, into ah
mov ah, 09h
get the quote and store it into dx
mov dx, offset Intro
print the store info
int 21h
store 01h into ah
mov ah,01h
get user input
int 21h
subtract 30h to user input
sub,al,30h
mov dx, offset prompt
set 0ah into bl
mov bl,0ah
multply the bl reg to the al reg
mul al,bl
get second number input
mov ah, 01h


accept the user input
int 21h
subtract 30h from the user input
sub al,30h
add the first input with the second input
add bl,al
mov cl,bl
mov bl,cl
put 09h into the reg
mov ah,09h
move quote into mem
mov dx, offset Hello
print it out
int 21h
get the first digit to print first
mov dx, offset tens
print it out
int 21h
get second digit to dx
mov dx, offset one
print it out
int 21h
increment the number by 1
inc byte ptr ones



Lastly for the flow chart was the one that takes the user input, along with incrementing and title.

Conclusion:

In conclusion this lab was both time consuming and kind of tedious, abut at the same time it does help you learn about the assembly language debugger. It helps me understand how machine language come into play with hex, binary and from assembly instruction. Working with how the problem runs and how they should be executed. Using the debugger was great way to practice and understand what was going on during the execution of the code.