EEE174 –CPE185 INTRODUCTION TO MICROPROCESSORS

LAB 1 PART 1 & 2

Lab Session: Wednesday 6:30PM - 9:10PM

Section 32385

Lab Instructor: Sean Kennedy

Student Name: Andrew Robertson

TABLE OF CONTENTS

Part 1 3
Overview 3
Pre-Lab 4
Problem Definition: 4
Flow Chart: 4
Lab Discussion 5
Work Performed / Solution: 5
Listing Files(s): 9
Part 2 12
Overview 12
Pre-Lab 13
Program restriction: 13
Flow Chart: 14
Lab Discussion 15
Work Performed / Solution: 15
Listing Files(s): 19
Final Conclusion 31

PART 1

OVERVIEW

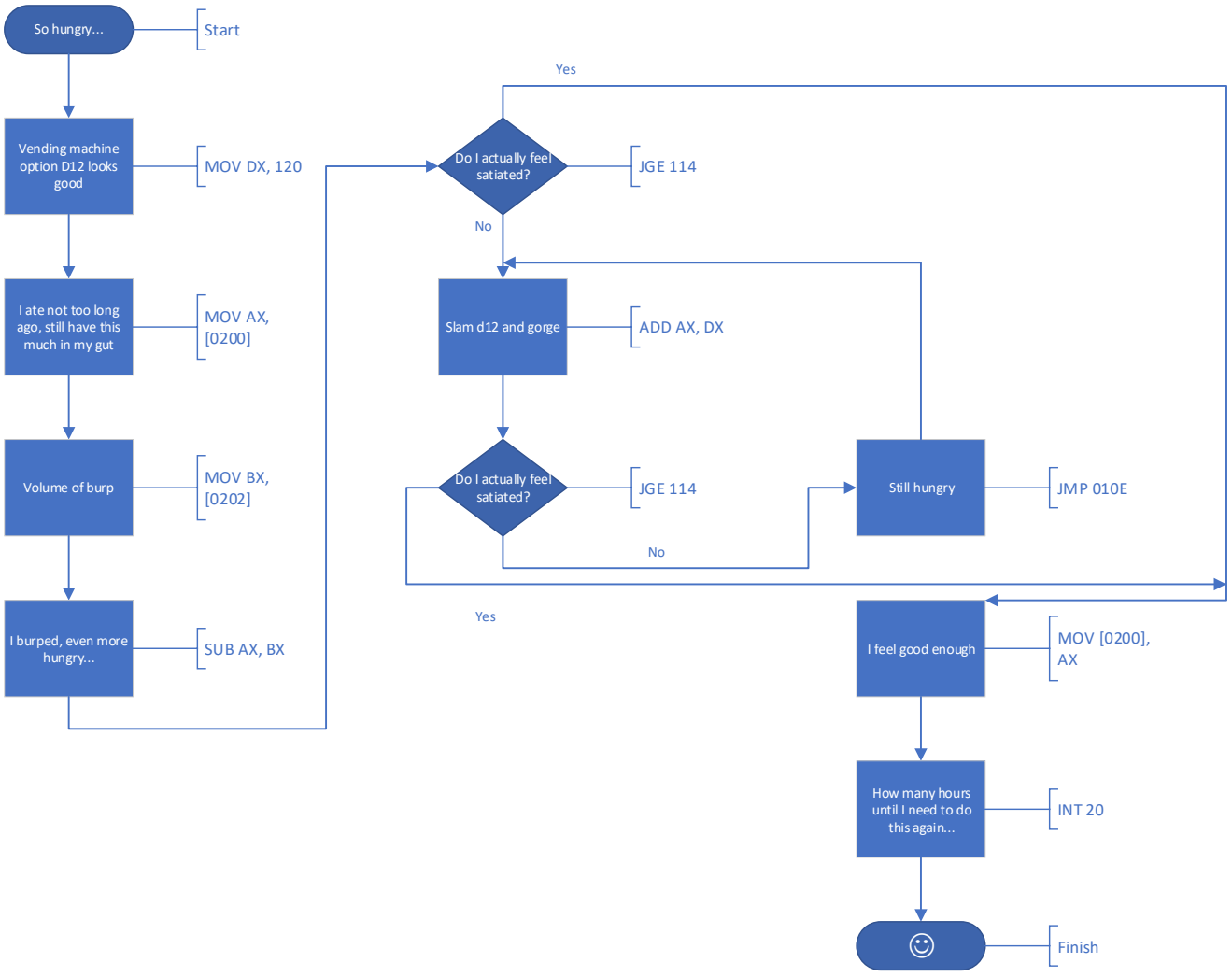
This labs purpose is to give us a brief overview of x86 workings and commands as well as provide simple exercises to help us remember some C programming syntax and style. We will use DOS debugger to edit memory locations, un-assemble hex code to view it, assemble code to its hex form, dump memory locations to view data, step through our program to view flags, and run the program entirely. After this, we create a C program to parallel the assembly program's functionality.

PRE-LAB

PROBLEM DEFINITION:

We are to use the provided code to view and trace its actions with hope to understand the workings of assembly programming and flags.

FLOW CHART:



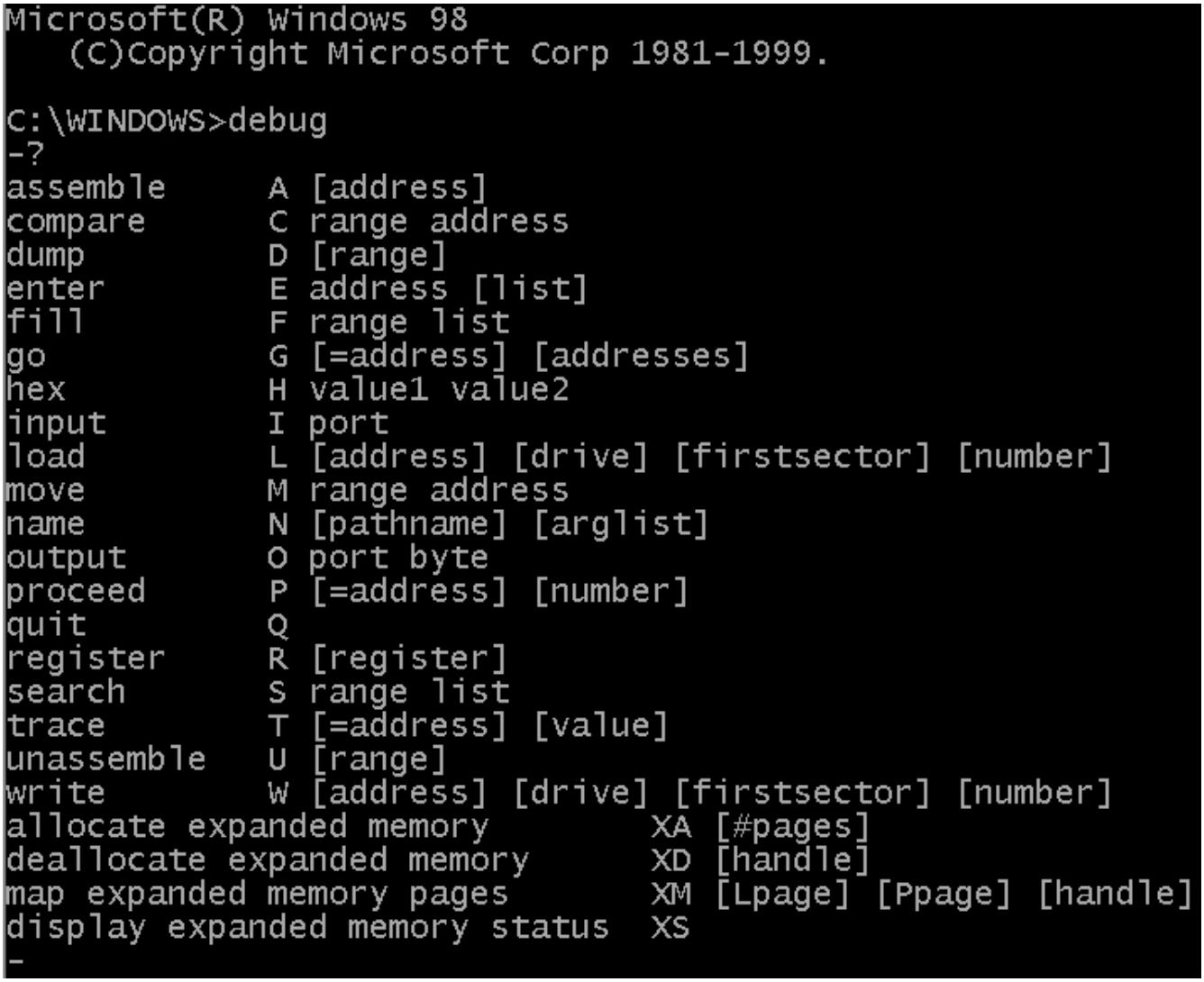
To make the flow of the program more relatable and easy to understand we were instructed to come up with a story to describe the programs actions. I described this program using a person standing in front of a vending machine trying to decide if he's hungry enough to get something from the machine. In my case, the value being monitored is a sort of rating for his / her stomach with zero and positive values being satiated and negative values being hungry.

LAB DISCUSSION

WORK PERFORMED / SOLUTION:

1)

After putting the Win98 Virtual machine file on my thumb drive I then launched the VMware program and selected the file as instructed. I was excited to launch this because I've run virtual machines before and love the idea or running one operating system inside of another with added tool to migrate information between them. The outcome was as expected, Win98 DOS debug inside of Windows 10.



Here, we see a full list of commands that can be given to the debug tool.

2)

Next, I entered three commands. D 100. D 100 110. D 100 200. This is the result:

```
-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   ...........</...
-d 100 110
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
-d 100 200
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                                                .
```

D 100 displays data and ascii interpretations starting at memory location 100 and going up to but not including 0x80 further.

D 100 to 110 works similarly but stops the display at memory location 110 and includes it.

D 100 to 200 is basically the same as the previous instruction but stops at and includes 200.

The addressing scheme is hexadecimal, displaying 0x10 or 16 bytes per line.

3)

I then used the e (enter) command to enter data into the CS segment so we can have the program desired in memory and ready to run. Using a text file for this step with newlines and spaces exactly where they would be entered as if typing in the console makes the process simply copy and paste (via the MSDOS menu). That way if there are any discrepancies it can be quickly changed and re-entered without having to go through the manual process.

4)

Next is checking the code entered to make sure it is as expected and tracing or running the code doesn't cause huge problems for the OS or hardware.

```
-u 100 118
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 7D03          JGE     0115
0F68:0112 EBFA          JMP     010E
0F68:0114 A30002        MOV     [0200],AX
0F68:0117 CD20          INT     20
-e 100
0F68:0100  BA.     20.     01.     A1.     00.     02.     8B.     1E.
0F68:0108  02.     02.     29.     D8.     7D.     06.     01.     D0.
0F68:0110  7D.     03.02   EB.     FA.     A3.     00.     02.     CD.
-u 100 118
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
```

My output was not as expected the first time. I entered 7D03 for JGE 3 bytes ahead instead of 7D02 for JGE 2 bytes ahead. I found out that if space it hit without entered data when using the e command, that segment is left untouched, so I only had to enter one value to fix it instead of the whole thing again.

5)

After verifying all is well we can now begin to trace the program. The instruction doesn't show it but the r (register modify) command can be used two ways. One way is the way shown, typing r alone gives all register information. Exactly what we would want for the start of a trace since the next command is always the one shown. The second way is what it asks us to achieve, typing the name of a register after r lets you modify its contents. For example, r ip lets me modify the instruction pointer so I can place it at the start of the program.

6)

This program references memory locations for data so to have a predictable running program we must put data there we want to use. If these locations were just to store data, then we wouldn't need to do this.

```
-e 200
0F68:0200  20.20   F9.01   75.50   04.02
-d 200 203
0F68:0200  20 01 50 02                                          .P.
-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=012
```

Here, I used the d (dump) command to verify the contents of the memory I just altered. After all was well it was ok to begin tracing through the program. All registers, including the status of the flags, are on display as well as the next instruction to be executed. Note when memory is referenced, its contents are also displayed as the total referenced value and not as the little-endian representation entered earlier.

7)

Last is to learn how the g (go) command works.

```
-g = 100 10E

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=010E   NV UP EI NG NZ NA PO CY
0F68:010E 01D0          ADD     AX,DX
-g

Program terminated normally
-g = 100

Program terminated normally
```

The first entry, g = 100 10E runs the program beginning at CS 100 and stopping at 10E then displaying the register contents like t would.

The second entry, g, runs the program from the IP's current location until termination then displays a message to verify the program ran completed successfully.

The third entry, g = 100, runs the program beginning at CS 100 until termination then displays a message to verify the program ran completed successfully.

8)

INT 20 exits the running program and returns control to DOS.

LISTING FILES(S):

Tracing chart

EEE 174												
Laboratory Exercise #1									Name: Andrew Robertson			
Program Tracing Chart												
			Registers:									
	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	000A	0130	MOV DX,0120
	0000	0000	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	0103	000A	0130	MOV AX, [0200]
	000A	0000	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	0106	000A	0130	MOV BX, [0202]
	000A	0130	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	010A	000A	0130	SUB AX, BX
	FEDA	0130	0000	0120	NV (0)	NZ (0)	NG (1)	0F68	010C	000A	0130	JGE 0114
	FEDA	0130	0000	0120	NV (0)	NZ (0)	NG (1)	0F68	010E	000A	0130	ADD AX, DX
	FFFA	0130	0000	0120	NV (0)	NZ (0)	NG (1)	0F68	0110	000A	0130	JGE 0114
	FFFA	0130	0000	0120	NV (0)	NZ (0)	NG (1)	0F68	0112	000A	0130	JMP 010E
	FFFA	0130	0000	0120	NV (0)	NZ (0)	NG (1)	0F68	010E	000A	0130	ADD AX, DX
	011A	0130	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	0110	000A	0130	JGE 0114
	011A	0130	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	0114	000A	0130	MOV [0200],AX
	011A	0130	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	0117	011A	0130	INT 20
	0000	0000	0000	0000	NV(0)	NZ(0)	PL(0)	0F68	0100	0002	0001	MOV DX,0120
	0000	0000	0000	0120	NV(0)	NZ(0)	PL(0)	0F68	0103	0002	0001	MOV AX, [0200]
	0002	0000	0000	0120	NV(0)	NZ(0)	PL(0)	0F68	0106	0002	0001	MOV BX, [0202]
	0002	0001	0000	0120	NV(0)	NZ(0)	PL(0)	0F68	010A	0002	0001	SUB AX, BX
	0001	0001	0000	0120	NV(0)	NZ(0)	PL(0)	0F68	010C	0002	0001	JGE 0114
	0001	0001	0000	0120	NV(0)	NZ(0)	PL(0)	0F68	0114	0002	0001	MOV [0200],AX
	0001	0001	0000	0120	NV(0)	NZ(0)	PL(0)	0F68	0117	0001	0001	INT 20

Hand assembly

EEE 174
Laboratory Hand-Assembly Template
Dahlquist/Stoffers/Schultz

Instruction: MOV DX,0120

Address:	CS	**100**	Operation:	MOV	Dest.	DX	Source: **120**
		Immediate to register					
Instruction Format		1011 w reg:immediate data					
		(w = 1)	(reg = 010)	little endian			
Binary:	1011	1010	2001				
Hex:	B	A	2001	Can check via instructions			

Instruction: MOV AX,[0200]

Address:	CS	: 0103	Operation:	MOV	Dest.	AX	**[0200]**
		memory to AX					
Instruction Format		1010 000w : full displacement					
		(w=1)					
Binary:	1010	0001	0002				
Hex:	A10002						

Instruction: MOV BX,[0202]

Address:	CS	**106**	Operation:	MOV	Dest.	BX	Source: [0202]
		memory to register					
Instruction Format		1000 101w mod:reg r/m					
		w = 1	mod =00	reg =011	r/m =110		
Binary:	1000	1011	0001	1110	0202		
Hex:	8 B 1 E 02 02						

Instruction: SUB AX,BX

Address:	CS	: 10A	Operation:	SUB	Dest.	AX	Source: **BX**
	reg1 to reg2						
Instruction Format		0010 100w : 11:reg1 reg2					
		w =1	reg1 = 011		reg2 = 000		
Binary:	0010	1001	1101	1000			
Hex:	29D8						

Instruction: JGE 0114

		jump if condition is met					
Address:	CS	: 10C	Operation:	JGE	Dest.	0114	Source:
Instruction Format		0111 tttn 8 bit displacement (distance from current instruction)					
			ttn = 1101				
Binary:	0111	1101	0110				
Hex:	7 D 06						

C program to add two numbers

```cpp
#include <iostream>
using namespace std;

int addTwo(int x,int y){
    return x+y;
}

int main() {
    int y;
    int a;
    int b;

    cout << "Please enter first number: \n";
    cin >> a;
    cout << "Please enter second number: \n";
    cin >> b;

    y = addTwo(a, b);
    printf("Result: %d", y);
    return 0;
}
```

C program to output Hello to standard out

```cpp
#include <iostream>
using namespace std;

int main() {
    printf("Hello");
    return 0;
}
```

C program to mimic the MASM program

```cpp
#include<stdio.h>

int main() {
   int debit = 0x10;
   int cost = 0x40;
   int AH, AL, BL, DH, DL;

   DH = 0x120;
   AH = debit;
   AL = cost;
   AH -= AL;

   while(AH < 0) AH += DH;
   debit = AH;
   BL = DL;

   printf("%d \n",debit);
}
```

Commented MASM program

```
MOV DX,0120     ;Copy immediate value 0x120 into 16 bit reg DX
MOV AX,[0200]   ;Copy value stored in memory location 0200 to 16 bit reg AX
MOV BX,[0202]   ;Copy value stored in memory location 0202 to 16 bit reg BX
SUB AX,BX       ;Subtract BX from AX and store result in AX.
JGE 0114        ;If AX is positive, copy immediate value 0114 to IP
ADD AX,DX       ;ADD DX to AX then store result in AX
JGE 0114        ;If AX is positive, copy immediate value 0114 to IP
JMP 010E        ;Unconditoinally copy immediate value 010E to IP
MOV [0200],AX   ;Copy value stored in AX to memory location 0200
INT 20          ;Terminate program
```

PART 2

OVERVIEW

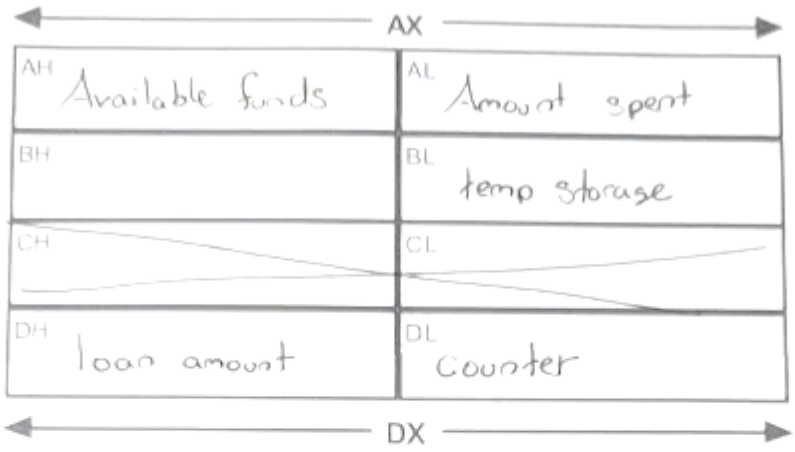
We will take the program examined in step one and modify it in a few ways. First, we are given a register we are unable to use. Second, we are given a block of memory we can use. Third, we must implement the program as an 8-bit instruction set (for example, AX becomes AL and AH). Fourth, we can only use one conditional jump. And last, we are to implement a counter to keep track of the number of times the loan or bailout is given and display it after displaying our names and assignment title.

PRE-LAB

PROGRAM RESTRICTION:

The register I am unable to use is CX and the memory I am given begins at DS:0204.

Registers Used:

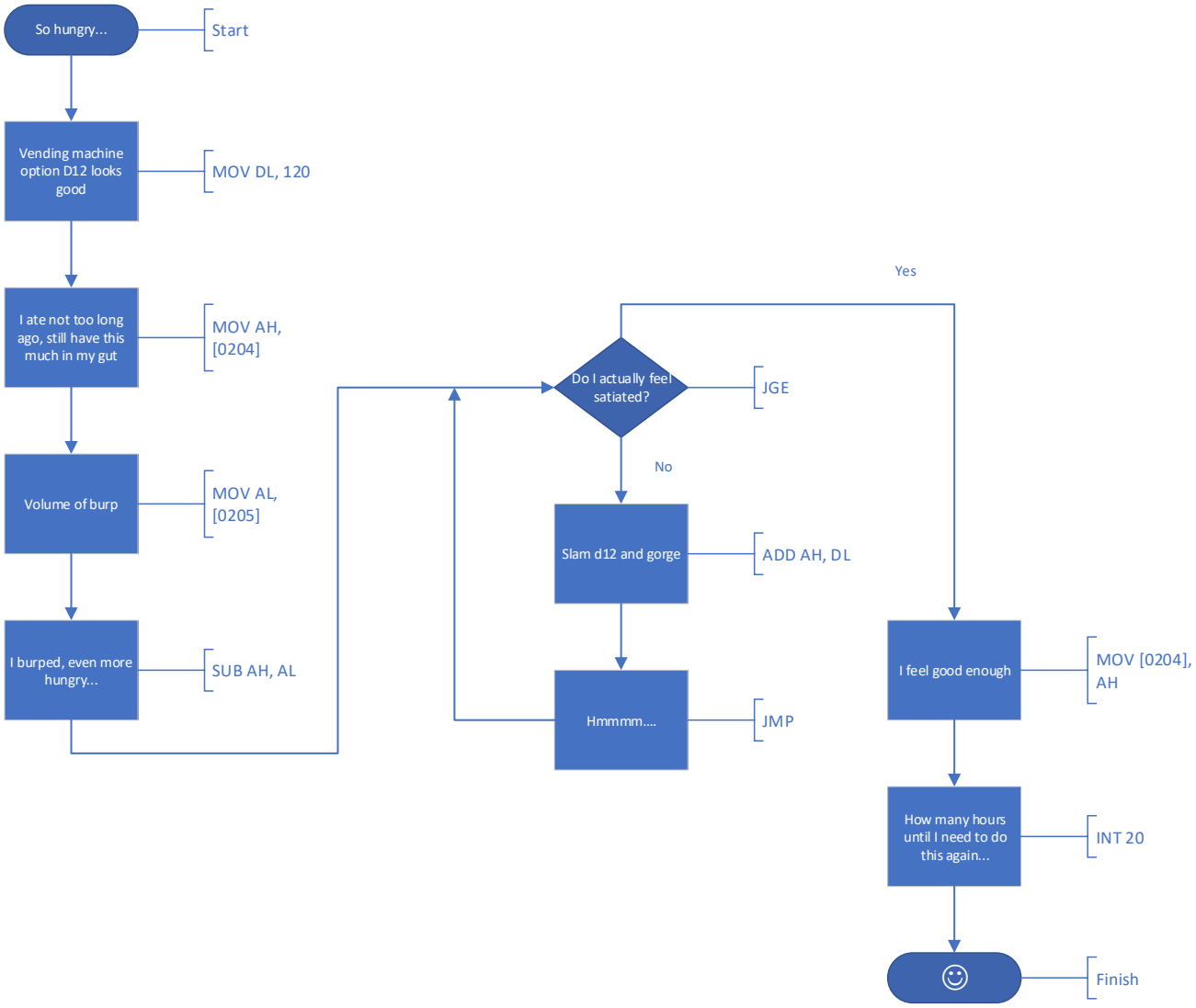


Memory Locations:

Address:	204	205	206	22F
Label:	debit	cost	start_msg	end_msg
Contents:	Starting funds	amount spent	Name assignment	counter message
			String	String

Flow Chart:

New flowchart depicting the design using only one conditional jump. Since the old design asked the same JGE question twice to examine the same data, this does not affect the outcome of the program at all.



LAB DISCUSSION

WORK PERFORMED / SOLUTION:

1)

After I had an idea of the code I wanted to implement I then preceded to hand assemble the instructions, so they could be entered in to memory and verified. This was not something that worked the very first try either. I had two issues on the first attempt. I counted the byte displacement of my unconditional jump 2 bytes short and needed to fix that. I also needed to fix my counter increment instruction for a reason I did not expect. It was entered correctly but interpreted incorrectly, I learned that the alternate encoding for the increment instruction of a register references the 16 bit register and not the 8 bit version. Although this still would have worked fine for a reasonable number of counts it was unacceptable due to the limitation of 8 bit operations only and so the original encoding needed to be used so I could set w = 0 and reference the 8 bit DL vs 16 bit DX.

This is the result:

```
0F68:0110  7D.00   02.A0   EB.05   FA.02   A3.28   00.C4   02.7D   CD.06
-
-e 118
0F68:0118  20.FE   3E.C2   43.00   04.F4   34.EB   00.F8   57.88   0F.26
-
-e 120
0F68:0120  BA.04   42.02   86.88   E9.D3   65.BA   FE.2F   BF.02   81.B4
-
-e 128
0F68:0128  00.09   8B.CD   36.21   92.B4   DE.02   8B.88   44.DA   FE.80
-
-e 130
0F68:0130  BE.C2   C6.30   DB.CD   8B.21   74.CD   09.20
-e 204
0F68:0204  FE.10
-
-e 205
0F68:0205  06.40
-
-e 206 "Andrew Robertson, x86 Lab Pt.2" 0d 0a "$"
-
-e 22F "Loan distributions: $"
-
-u 100 135
0F68:0100 C7C20602      MOV     DX,0206
0F68:0104 C6C409        MOV     AH,09
0F68:0107 CD21          INT     21
0F68:0109 B620          MOV     DH,20
0F68:010B 8A260402      MOV     AH,[0204]
0F68:010F B200          MOV     DL,00
0F68:0111 A00502        MOV     AL,[0205]
0F68:0114 28C4          SUB     AH,AL
0F68:0116 7D06          JGE     011E
0F68:0118 FEC2          INC     DL
0F68:011A 00F4          ADD     AH,DH
0F68:011C EBF8          JMP     0116
0F68:011E 88260402      MOV     [0204],AH
0F68:0122 88D3          MOV     BL,DL
0F68:0124 BA2F02        MOV     DX,022F
0F68:0127 B409          MOV     AH,09
0F68:0129 CD21          INT     21
0F68:012B B402          MOV     AH,02
0F68:012D 88DA          MOV     DL,BL
0F68:012F 80C230        ADD     DL,30
0F68:0132 CD21          INT     21
0F68:0134 CD20          INT     20
-
-d 206 250
0F68:0200                   41 6E-64 72 65 77 20 52 6F 62        Andrew Rob
0F68:0210  65 72 74 73 6F 6E 2C 20-78 38 36 20 4C 61 62 20   ertson, x86 Lab
0F68:0220  50 74 2E 32 0D 0A 24 0D-75 02 88 04 89 36 E3 4C   Pt.2..$.u....6.L
0F68:0230  6F 61 6E 20 64 69 73 74-72 69 62 75 74 69 6F 6E   oan distribution
0F68:0240  73 3A 20 24 03 00 3C 0D-C3 AC E8 04 F9 75 04 3C   s: $..<......u.<
0F68:0250  3B                                                ;
```

2)

Now it's time to trace the program and make sure it is operating correctly. To avoid redundancy I'll display the key pieces of the raw trace here and place the tracing chart in the listing files area at the end of this document.

```
-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 C7C20602      MOV     DX,0206
-t

AX=0000  BX=0000  CX=0000  DX=0206  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0104   NV UP EI PL NZ NA PO NC
0F68:0104 C6C409        MOV     AH,09
-t

AX=0900  BX=0000  CX=0000  DX=0206  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0107   NV UP EI PL NZ NA PO NC
0F68:0107 CD21          INT     21
```

This snippet is what allows this program to display a string. The first MOV instruction loads DX with the value of the memory location where the string is located. This needs to be done because INT 21 when AH is 9 will only look to DX for this location. The second MOV is to tell the INT 21 instruction what operation it is to perform, the interrupt will only look at AH for this value. The result is to display a string located with the starting address of 0206.

```
AX=F040  BX=0000  CX=0000  DX=2001  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0116   NV UP EI NG NZ NA PE NC
0F68:0116 7D06        JGE     011E
-t

AX=F040  BX=0000  CX=0000  DX=2001  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0118   NV UP EI NG NZ NA PE NC
0F68:0118 FEC2        INC     DL
-t

AX=F040  BX=0000  CX=0000  DX=2002  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=011A   NV UP EI PL NZ NA PO NC
0F68:011A 00F4        ADD     AH,DH
-t

AX=1040  BX=0000  CX=0000  DX=2002  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=011C   NV UP EI PL NZ NA PO CY
0F68:011C EBF8        JMP     0116
-t
```

I also think this segment is important because it is a perfect example of how deliberate the order of instruction must be. The jump at the beginning here will direct the IP pointer to 011E if AH turns out to be positive in this case but more accurately it will check the sign and overflow flags to make sure they are equal. If the add instruction that is next was instead before the increment instruction this would not work as intended. The flags the JGE would examine would be the result of the increment instruction and not the add, therefore we would be evaluating whether the count is non-negative instead of the value in AH. This would cause the JGE to evaluate true every time this snippet was executed.

```
-g
Andrew Robertson, x86 Lab Pt.2
Loan distributions: 2
Program terminated normally
```

Here, I simply wanted to show what the console output looks like when the go command is given for this program.

3)

Last, the easy part (for me) is to create a C style program to parallel the MASM program

```
;C Style representation     ;C Style representation     ;C Style representation     ;C Style representation

#include<stdio.h>

int main() {
  char* start_msg = "Andrew Robertson, x86 Lab Pt.2";
  char* end_msg = "Loan distributions: ";
  int debit = 0x10;
  int cost = 0x40;
  int AH, AL, BL, DH, DL;

  printf("%s \n",start_msg);

  DH = 0x20;
  AH = debit;
  DL = 0;
  AL = cost;
  AH -= AL;

  while(AH < 0){
    DL++;
    AH += DH;
  }
  debit = AH;
  BL = DL;

  printf("%s",end_msg);
  DL = BL;
  printf("%d \n",DL);
}

;console output
Andrew Robertson, x86 Lab Pt.2
Loan distributions: 2
```

Note, the pointers created here and subsequently the temporary storage of BL are not necessary. They were only created to mimic the assembly program. For instance, both the printf commands entered that have a char* passed to them could have simply used the string in their initial argument instead.

There was also a question on how inline assembly code would look, I found what I think would be a good resource: https://www.codeproject.com/Articles/15971/Using-Inline-Assembly-in-C-C . I attempted to follow some of what was done but to no avail. I understand that inline would be creating a function entirely in assembly to be run exactly the way you would like it to but had a really hard time with the syntax and styling under such a short time. Accessing external variables is certainly interesting. Knowing I'll need this skill at some point relatively soon I would like to give it a lot more time.

LISTING FILES(S):

EEE 174														
Laboratory Exercise #1 part 2									Name: Andrew Robertson					
Program Tracing Chart														
			Registers:											
	AX:	BX:	CX:	DX:	OF:	ZF:	SF:	CS:	IP:	DS:204	DS:205	DS:206	DS:22F	Next Instruction:
Value:--->	0000	0000	0000	0000	NV(0)	NZ(0)	PL(0)	0F68	100	10	40	srt_msg	end_msg	MOV DX, 0206
	0000	0000	0000	0206	NV(0)	NZ(0)	PL(0)	0F68	104	10	40	srt_msg	end_msg	MOV AH, 09
	0900	0000	0000	0206	NV(0)	NZ(0)	PL(0)	0F68	107	10	40	srt_msg	end_msg	INT 21
	0900	0000	0000	0206	NV(0)	NZ(0)	PL(0)	0F68	109	10	40	srt_msg	end_msg	MOV DH, 20
	0900	0000	0000	2006	NV(0)	NZ(0)	PL(0)	0F68	10B	10	40	srt_msg	end_msg	MOV AH, [0204]
	1000	0000	0000	2006	NV(0)	NZ(0)	PL(0)	0F68	10F	10	40	srt_msg	end_msg	MOV DL,00
	1000	0000	0000	2000	NV(0)	NZ(0)	PL(0)	0F68	111	10	40	srt_msg	end_msg	MOV AL, [0205]
	1040	0000	0000	2000	NV(0)	NZ(0)	PL(0)	0F68	114	10	40	srt_msg	end_msg	SUB AH, AL
	D040	0000	0000	2000	NV(0)	NZ(0)	NG(1)	0F68	116	10	40	srt_msg	end_msg	JGE 011E
	D040	0000	0000	2000	NV(0)	NZ(0)	NG(1)	0F68	118	10	40	srt_msg	end_msg	INC DL
	D040	0000	0000	2001	NV(0)	NZ(0)	PL(0)	0F68	11A	10	40	srt_msg	end_msg	ADD AH, DH
	F040	0000	0000	2001	NV(0)	NZ(0)	NG(1)	0F68	11C	10	40	srt_msg	end_msg	JMP 0116
	F040	0000	0000	2001	NV(0)	NZ(0)	NG(1)	0F68	116	10	40	srt_msg	end_msg	JGE 011E
	F040	0000	0000	2001	NV(0)	NZ(0)	NG(1)	0F68	118	10	40	srt_msg	end_msg	INC DL
	F040	0000	0000	2002	NV(0)	NZ(0)	PL(0)	0F68	11A	10	40	srt_msg	end_msg	ADD AH, DH
	1040	0000	0000	2002	NV(0)	NZ(0)	PL(0)	0F68	11C	10	40	srt_msg	end_msg	JMP 0116
	1040	0000	0000	2002	NV(0)	NZ(0)	PL(0)	0F68	116	10	40	srt_msg	end_msg	JGE 011E
	1040	0000	0000	2002	NV(0)	NZ(0)	PL(0)	0F68	11E	10	40	srt_msg	end_msg	MOV [0204]
	1040	0000	0000	2002	NV(0)	NZ(0)	PL(0)	0F68	122	10	40	srt_msg	end_msg	MOV BL, DL
	1040	0002	0000	2002	NV(0)	NZ(0)	PL(0)	0F68	124	10	40	srt_msg	end_msg	MOV DX, 022F
	1040	0002	0000	022F	NV(0)	NZ(0)	PL(0)	0F68	127	10	40	srt_msg	end_msg	MOV AH, 09
	0940	0002	0000	022F	NV(0)	NZ(0)	PL(0)	0F68	129	10	40	srt_msg	end_msg	INT 21
	0940	0002	0000	022F	NV(0)	NZ(0)	PL(0)	0F68	12B	10	40	srt_msg	end_msg	MOV AH, 02
	0240	0002	0000	022F	NV(0)	NZ(0)	PL(0)	0F68	12D	10	40	srt_msg	end_msg	MOV DL, BL
	0240	0002	0000	202	NV(0)	NZ(0)	PL(0)	0F68	12F	10	40	srt_msg	end_msg	ADD DL, 30
	0240	0002	0000	232	NV(0)	NZ(0)	PL(0)	0F68	132	10	40	srt_msg	end_msg	INT 21
	0240	0002	0000	232	NV(0)	NZ(0)	PL(0)	0F68	134	10	40	srt_msg	end_msg	INT 20

	AX:	BX:	CX:	DX:	OF:	ZF:	SF:	CS:	IP:	DS:204	DS:205	DS:206	DS:22F	Next Instruction:
	0000	0000	0000	0000	NV(0)	NZ(0)	PL(0)	0F68	0100	40	10	srt_msg	end_msg	MOV DX, 0206
	0000	0000	0000	0206	NV(0)	NZ(0)	PL(0)	0F68	0104	40	10	srt_msg	end_msg	MOV AH, 09
	0900	0000	0000	0206	NV(0)	NZ(0)	PL(0)	0F68	0107	40	10	srt_msg	end_msg	INT 21
	0900	0000	0000	0206	NV(0)	NZ(0)	PL(0)	0F68	0109	40	10	srt_msg	end_msg	MOV DH, 20
	0900	0000	0000	2006	NV(0)	NZ(0)	PL(0)	0F68	010B	40	10	srt_msg	end_msg	MOV AH, [0204]
	4000	0000	0000	2006	NV(0)	NZ(0)	PL(0)	0F68	010F	40	10	srt_msg	end_msg	MOV DL,00
	4000	0000	0000	2000	NV(0)	NZ(0)	PL(0)	0F68	0111	40	10	srt_msg	end_msg	MOV AL, [0205]
	4000	0000	0000	2000	NV(0)	NZ(0)	PL(0)	0F68	0114	40	10	srt_msg	end_msg	SUB AH, AL
	3010	0000	0000	2000	NV(0)	NZ(0)	PL(0)	0F68	0116	40	10	srt_msg	end_msg	JGE 011E
	3010	0000	0000	2000	NV(0)	NZ(0)	PL(0)	0F68	011E	40	10	srt_msg	end_msg	MOV [0204],AH
	3010	0000	0000	2000	NV(0)	NZ(0)	PL(0)	0F68	0122	30	10	srt_msg	end_msg	MOV BL, DL
	3010	0000	0000	2000	NV(0)	NZ(0)	PL(0)	0F68	0124	30	10	srt_msg	end_msg	MOV DX, 022F
	3010	0000	0000	022F	NV(0)	NZ(0)	PL(0)	0F68	0127	30	10	srt_msg	end_msg	MOV AH, 09
	0910	0000	0000	022F	NV(0)	NZ(0)	PL(0)	0F68	0129	30	10	srt_msg	end_msg	INT 21
	0910	0000	0000	022F	NV(0)	NZ(0)	PL(0)	0F68	012B	30	10	srt_msg	end_msg	MOV AH, 02
	0210	0000	0000	022F	NV(0)	NZ(0)	PL(0)	0F68	012D	30	10	srt_msg	end_msg	MOV DL, BL
	0210	0000	0000	0200	NV(0)	NZ(0)	PL(0)	0F68	012F	30	10	srt_msg	end_msg	ADD DL, 30
	0210	0000	0000	0230	NV(0)	NZ(0)	PL(0)	0F68	0132	30	10	srt_msg	end_msg	INT 21
	0210	0000	0000	0230	NV(0)	NZ(0)	PL(0)	0F68	0134	30	10	srt_msg	end_msg	INT 20

EEE 174

Laboratory Hand-Assembly Template

Dahlquist/Stoffers/Schultz

Instruction: MOV DX,206

Address:	CS	**100**		Operation:	MOV	Dest.:	DX	Source:	**206**
		immediate to reg							
Instruction Format		1100 011w 11 000 reg : immediate data							
Binary:	1100	0111	1100	0010	0602				
Hex:	C7 C2 06 02								

Instruction: MOV AH,09

Address:	CS	**104**		Operation:	MOV	Dest.:	AH	Source:	**9**
		immediate to reg							
Instruction Format		1100 011w 11 000 reg : immediate data							
Binary:	1100	0110	1100	0100	0009				
Hex:	C6 C4 09								

Instruction: INT 21

Address:	CS	**107**		Operation:	INT	Dest.:		Source:	
			interrupt						
Instruction Format		1100 1101 : type							
Binary:	1100	1101	0021						
Hex:	CD 21			Can check via instructions					

Instruction: MOV DH,20

Address:	CS	**109**		Operation:	MOV	Dest.:	DH	Source:	**20**
		Immediate to register							
Instruction Format		1011 w reg:immediate data							
Binary:	1011	0110	0020						
Hex:	B6 20			Can check via instructions					

Instruction: MOV AH, [0204]

Address:	CS	**10B**		Operation:	MOV	Dest.:	AH	Source:	**[0204]**
		mem to register							
Instruction Format		1000 101w:mod reg r/m							
	w=0	mod=00		reg=100		r/m=110			
Binary:	1000	1010	0010	0110	0204				
Hex:	8A 26 04 02								

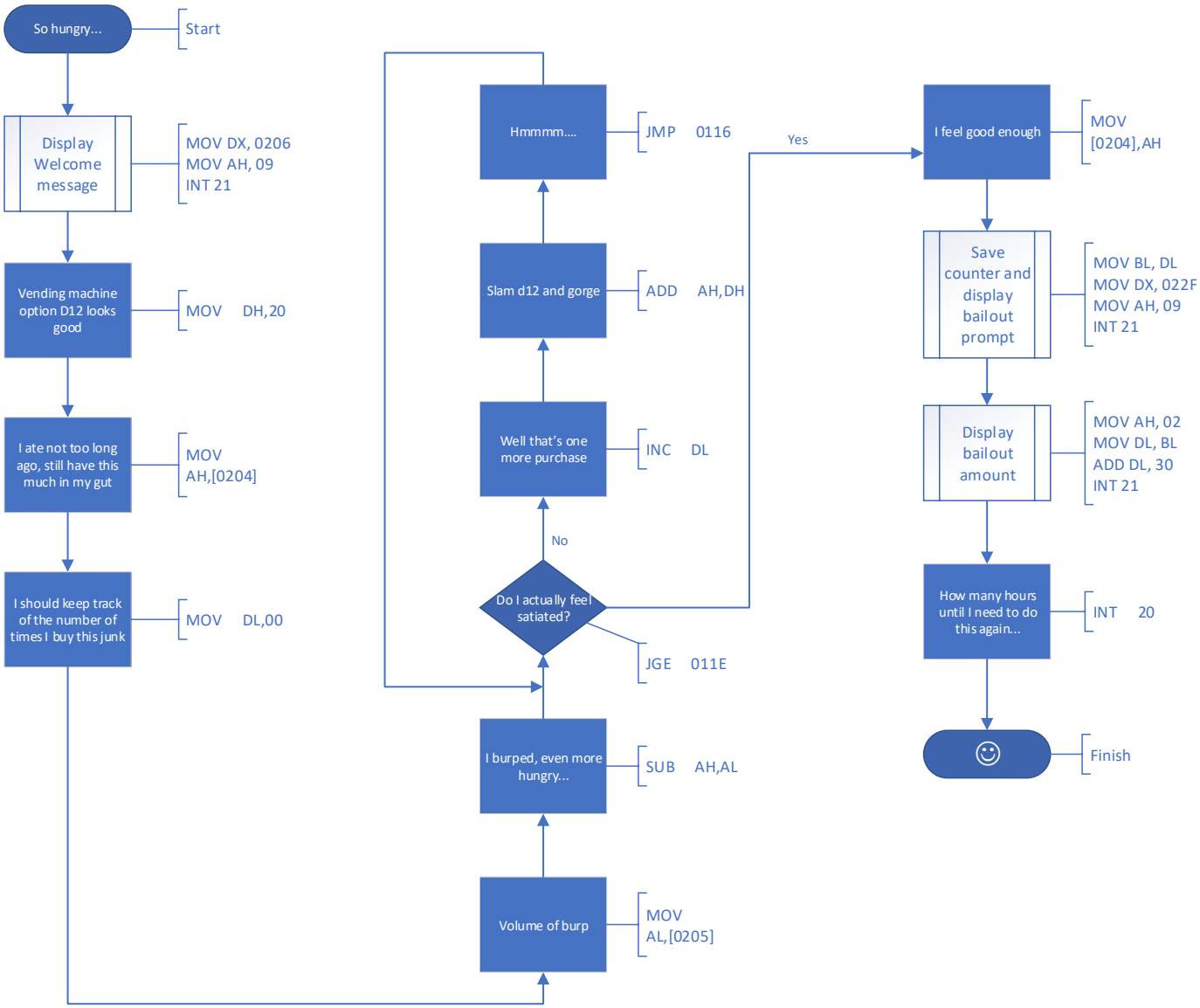
Instruction: MOV DL, 00

Address:	CS	**10F**		Operation:	MOV	Dest.:	DH	Source:	**0**
		immediate to register							
Instruction Format		1011 w reg immediate data							
Binary:	1011	0010	0000						
Hex:	B2 00								

Instruction	MOV AL, [0205]									
Address:	CS	**111**		Operation:	MOV	Dest.:	AL	Source:	**[0205]**	
		mem to AL								
Instruction Format		1010 000w : full displacement								
Binary:	1010	0000	0502							
Hex:	A0 05 02									
Instruction	SUB AH, AL									
Address:	CS	**114**		Operation:	SUB	Dest.:	AH	Source:	**AL**	
		reg to reg								
Instruction Format		0010 100w : 11 reg1 reg2								
Binary:	0010	1000	1100	0100						
Hex:	28 C4									
Instruction	JGE 11E									
			jump if condition is met							
Address:	CS	**116**		Operation:	JGE	Dest.:	11E	Source:		
Instruction Format		0111 tttn : full displacement								
Binary:	0111	1101	(distance)			this will be 2 bits wide? Count from end of this instructio				
Hex:	7D 06									
Instruction	INC DL									
Address:	CS	**118**		Operation:	INC	Dest.:	DH	Source:		
Instruction Format		1111 111w 11 000 reg								
Binary:	1111	1110	1100	0010		(Alternate encoding produces INC DX instead...)				
Hex:	FE C2									
Instruction	ADD AH, DH									
Address:	CS	**11A**		Operation:	ADD	Dest.:	AH	Source:	**DL**	
		reg to reg								
Instruction Format		0000 000w : 11 reg1 reg2								
Binary:	0000	0000	1111	0100						
Hex:	00 F4									
Instruction	JMP 116									
Address:	CS	**11C**		Operation:	JMP	Dest.:	116	Source:		
Instruction Format		1110 1011 : 8 bit displacement								
Binary:	1110	1011	1111	1001						
Hex:	EB F8									

Instruction	MOV [0204], AH									
Address:	CS	**11E**		Operation:	MOV		Dest.:	[0204]	Source:	**AH**
		reg to mem								
Instruction Format		1000 100w : mod reg r/m								
Binary:	1000	1000	0010	0110	0204					
Hex:	88 26 04 02									
Instruction:	MOV BL, DL									
Address:	CS	**122**		Operation:	MOV		Dest.:	BL	Source:	**DL**
		reg to reg								
Instruction Format		1000 100w 11 reg1 reg2								
Binary:	1000	1000	1101	0011						
Hex:	88 D3									
Instruction:	MOV DX, 022F									
Address:	CS	**124**		Operation:	MOV		Dest.:	DX	Source:	**022F**
		immediate to reg								
Instruction Format		1011 w reg immediate data								
Binary:	1011	1010	2F 02							
Hex:	BA 2F 02									
Instruction	MOV AH, 09									
Address:	CS	**: 127**		Operation:	MOV		Dest.:	AH	Source:	**9**
		immediate to reg								
Instruction Format		1011 w reg immediate data								
Binary:	1011	0100	0009							
Hex:	B4 09									
Instruction	INT 21									
Address:	CS	**129**		Operation:	INT		Dest.:		Source:	
		interrupt								
Instruction Format		1100 1101 type								
Binary:	1100	1101	0021							
Hex:	CD 21									
Instruction	MOV AH, 02									
Address:	CS	**12B**		Operation:	MOV		Dest.:	AH	Source:	**2**
		immediate to reg								
Instruction Format		1011 w reg immediate data								
Binary:	1011	0100	0002							
Hex:	B4 02									

Instruction:	MOV DL, BL								
Address:	CS	**12D**		Operation:	MOV	Dest.:	DL	Source:	**BL**
		reg to reg							
Instruction Format		1000 100w 11 reg1 reg2							
Binary:	1000	1000	1101	1010					
Hex:	88 DA								
Instruction:	ADD DL, 30								
Address:	CS	**12F**		Operation:	ADD	Dest.:	DL	Source:	**30**
		immediate to reg							
Instruction Format		1000 00sw 11 000 reg immediate							
Binary:	1000	0000	1100	0010	0030				
Hex:	80 C2 30								
Instruction	INT 21								
Address:	CS	**132**		Operation:	INT	Dest.:		Source:	
		interrupt							
Instruction Format		1100 1101 : type							
Binary:									
Hex:	CD 21								
Instruction	INT 20								
Address:	CS	**134**		Operation:	INT	Dest.:		Source:	
		interrupt							
Instruction Format		1100 1101 : type							
Binary:									
Hex:	CD 20								


So hungry...
Start
Display Welcome message
MOV DX, 0206
MOV AH, 09
INT 21
Vending machine option D12 looks good
MOV DH,20
I ate not too long ago, still have this much in my gut
MOV AH,[0204]
I should keep track of the number of times I buy this junk
MOV DL,00
Hmmmm....
JMP 0116
Slam d12 and gorge
ADD AH,DH
Well that's one more purchase
INC DL
No
Do I actually feel satiated?
JGE 011E
Yes
I burped, even more hungry...
SUB AH,AL
Volume of burp
MOV AL,[0205]
I feel good enough
MOV [0204],AH
Save counter and display bailout prompt
MOV BL, DL
MOV DX, 022F
MOV AH, 09
INT 21
Display bailout amount
MOV AH, 02
MOV DL, BL
ADD DL, 30
INT 21
How many hours until I need to do this again...
INT 20
Finish

```
;Hand assembled code

e 100
C7 C2 06 02 C6 C4 09 CD

e 108
21 B6 20 8A 26 04 02 B2

e 110
00 A0 05 02 28 C4 7D 06

e 118
FE C2 00 F4 EB F8 88 26

e 120
04 02 88 D3 BA 2F 02 B4

e 128
09 CD 21 B4 02 88 DA 80

e 130
C2 30 CD 21 CD 20


=============================================================================================================
;Config DS locations then check both CS and DS

e 204
10

e 205
40

e 206 "Andrew Robertson, x86 Lab Pt.2" 0d 0a "$"

e 22F "Loan distributions: $"

u 100 135

d 206 250

=============================================================================================================
;Code entered and checked

-e 100
0F68:0100  DE.C7   E8.C2   45.06   FA.02   AC.C6   AA.C4   3C.09   0D.CD
0F68:0108  75.21   FA.B6   56.20   8B.8A   36.26   92.04   DE.02   89.B2
0F68:0110  4C.00   FE.A0   5E.05   8E.02   06.28   08.C4   D3.7D   26.06
0F68:0118  80.FE   3E.C2   43.00   04.F4   34.EB   00.F8   57.88   0F.26
0F68:0120  BA.04   42.02   86.88   E9.D3   65.BA   FE.2F   BF.02   81.B4
0F68:0128  00.09   8B.CD   36.21   92.B4   DE.02   8B.88   44.DA   FE.80
0F68:0130  BE.C2   C6.30   DB.CD   8B.21   74.CD   09.20
-
-e 204
0F68:0204  FE.10
-
-e 205
0F68:0205  06.40
-
-e 206 "Andrew Robertson, x86 Lab Pt.2" 0d 0a "$"
-
-e 22F "Loan distributions: $"
-
-u 100 135
0F68:0100 C7C20602      MOV     DX,0206                 ;Copy value of the message string location into DX
0F68:0104 C6C409        MOV     AH,09                   ;Copy code for outputting string via INT 21 into AH
0F68:0107 CD21          INT     21                      ;Print stored string to standard out
0F68:0109 B620          MOV     DH,20                   ;Copy loan value of 20h into DH
0F68:010B 8A260402      MOV     AH,[0204]               ;Copy value stored in memory location 0204 to AH
```

```
0F68:010F B200        MOV     DL,00                   ;Copy counter value of 00h into DL
0F68:0111 A00502      MOV     AL,[0205]               ;Copy value stored in memory location 0205 to AL
0F68:0114 28C4        SUB     AH,AL                   ;Subtract Al from AH and store result to AH
0F68:0116 7D06        JGE     011E                    ;If AH is positive, jump to memory location 011E
0F68:0118 FEC2        INC     DL                      ;AH is negative, increment loan counter first...
0F68:011A 00F4        ADD     AH,DH                   ;...then provide loan, this order will provide accurate flags for JGE
0F68:011C EBF8        JMP     0116                    ;jump to JGE to check if AH is positive now
0F68:011E 88260402    MOV     [0204],AH               ;AH is positive, store this value in memory location 0204
0F68:0122 88D3        MOV     BL,DL                   ;Save the counter value before loading message location into DX
0F68:0124 BA2F02      MOV     DX,022F                 ;Copy value of the message string location into DX
0F68:0127 B409        MOV     AH,09                   ;Copy code for outputting string via INT 21 into AH
0F68:0129 CD21        INT     21                      ;Print stored string to standard out
0F68:012B B402        MOV     AH,02                   ;Copy code for outputting single character into AH for INT 21
0F68:012D 88DA        MOV     DL,BL                   ;Copy counter value back into DL to be output by INT 21
0F68:012F 80C230      ADD     DL,30                   ;Convert counter value to its ASCII representation (will only work for 0-9)
0F68:0132 CD21        INT     21                      ;Display single character
0F68:0134 CD20        INT     20                      ;Terminate program

-d 206 250
0F68:0200              41 6E-64 72 65 77 20 52 6F 62       Andrew Rob
0F68:0210  65 72 74 73 6F 6E 2C 20-78 38 36 20 4C 61 62 20   ertson, x86 Lab
0F68:0220  50 74 2E 32 0D 0A 24 0D-75 02 88 04 89 36 E3 4C   Pt.2..$.u....6.L
0F68:0230  6F 61 6E 20 64 69 73 74-72 69 62 75 74 69 6F 6E   oan distribution
0F68:0240  73 3A 20 24 03 00 3C 0D-C3 AC E8 04 F9 75 04 3C   s: $..<......u.<
0F68:0250  3B
```

==========

;Multi-loop trace ;Multi-loop trace ;Multi-loop trace ;Multi-loop trace ;Multi-loop trace

```
-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 C7C20602      MOV     DX,0206
-t

AX=0000  BX=0000  CX=0000  DX=0206  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0104   NV UP EI PL NZ NA PO NC
0F68:0104 C6C409        MOV     AH,09
-t

AX=0900  BX=0000  CX=0000  DX=0206  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0107   NV UP EI PL NZ NA PO NC
0F68:0107 CD21        INT     21
-r ip
IP 0107
:109
-r
AX=0900  BX=0000  CX=0000  DX=0206  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0109   NV UP EI PL NZ NA PO NC
0F68:0109 B620        MOV     DH,20
-t

AX=0900  BX=0000  CX=0000  DX=2006  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=010B   NV UP EI PL NZ NA PO NC
0F68:010B 8A260402      MOV     AH,[0204]                   DS:0204=10
-t

AX=1000  BX=0000  CX=0000  DX=2006  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=010F   NV UP EI PL NZ NA PO NC
0F68:010F B200        MOV     DL,00
-t

AX=1000  BX=0000  CX=0000  DX=2000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0111   NV UP EI PL NZ NA PO NC
0F68:0111 A00502        MOV     AL,[0205]                   DS:0205=40
-t
```

```
AX=1040  BX=0000  CX=0000  DX=2000  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 28C4        SUB     AH,AL
-t

AX=D040  BX=0000  CX=0000  DX=2000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0116   NV UP EI NG NZ NA PO CY
0F68:0116 7D06        JGE     011E
-t

AX=D040  BX=0000  CX=0000  DX=2000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0118   NV UP EI NG NZ NA PO CY
0F68:0118 FEC2        INC     DL
-t

AX=D040  BX=0000  CX=0000  DX=2001  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=011A   NV UP EI PL NZ NA PO CY
0F68:011A 00F4        ADD     AH,DH
-t

AX=F040  BX=0000  CX=0000  DX=2001  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=011C   NV UP EI NG NZ NA PE NC
0F68:011C EBF8        JMP     0116
-t

AX=F040  BX=0000  CX=0000  DX=2001  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0116   NV UP EI NG NZ NA PE NC
0F68:0116 7D06        JGE     011E
-t

AX=F040  BX=0000  CX=0000  DX=2001  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0118   NV UP EI NG NZ NA PE NC
0F68:0118 FEC2        INC     DL
-t

AX=F040  BX=0000  CX=0000  DX=2002  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=011A   NV UP EI PL NZ NA PO NC
0F68:011A 00F4        ADD     AH,DH
-t

AX=1040  BX=0000  CX=0000  DX=2002  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=011C   NV UP EI PL NZ NA PO CY
0F68:011C EBF8        JMP     0116
-t

AX=1040  BX=0000  CX=0000  DX=2002  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0116   NV UP EI PL NZ NA PO CY
0F68:0116 7D06        JGE     011E
-t

AX=1040  BX=0000  CX=0000  DX=2002  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=011E   NV UP EI PL NZ NA PO CY
0F68:011E 88260402      MOV     [0204],AH                   DS:0204=10
-t

AX=1040  BX=0000  CX=0000  DX=2002  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0122   NV UP EI PL NZ NA PO CY
0F68:0122 88D3        MOV     BL,DL
-t

AX=1040  BX=0002  CX=0000  DX=2002  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0124   NV UP EI PL NZ NA PO CY
0F68:0124 BA2F02      MOV     DX,022F
-t

AX=1040  BX=0002  CX=0000  DX=022F  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0127   NV UP EI PL NZ NA PO CY
0F68:0127 B409        MOV     AH,09
-t
```

```
AX=0940  BX=0002  CX=0000  DX=022F  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0129   NV UP EI PL NZ NA PO CY
0F68:0129 CD21        INT     21
-r ip
IP 0129
:12b
-r
AX=0940  BX=0002  CX=0000  DX=022F  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=012B   NV UP EI PL NZ NA PO CY
0F68:012B B402        MOV     AH,02
-t

AX=0240  BX=0002  CX=0000  DX=022F  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=012D   NV UP EI PL NZ NA PO CY
0F68:012D 88DA        MOV     DL,BL
-t

AX=0240  BX=0002  CX=0000  DX=0202  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=012F   NV UP EI PL NZ NA PO CY
0F68:012F 80C230      ADD     DL,30
-t

AX=0240  BX=0002  CX=0000  DX=0232  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0132   NV UP EI PL NZ NA PO NC
0F68:0132 CD21        INT     21
-r ip
IP 0132
:134
-r
AX=0240  BX=0002  CX=0000  DX=0232  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0134   NV UP EI PL NZ NA PO NC
0F68:0134 CD20        INT     20
-g

Program terminated normally

;console output

-g
Andrew Robertson, x86 Lab Pt.2
Loan distributions: 2
Program terminated normally


=============================================================================================================

;C Style representation      ;C Style representation      ;C Style representation      ;C Style representation

#include<stdio.h>

int main() {
  char* start_msg = "Andrew Robertson, x86 Lab Pt.2";
  char* end_msg = "Loan distributions: ";
  int debit = 0x10;
  int cost = 0x40;
  int AH, AL, BL, DH, DL;

  printf("%s \n",start_msg);

  DH = 0x20;
  AH = debit;
  DL = 0;
  AL = cost;
  AH -= AL;

  while(AH < 0){
     DL++;
     AH += DH;
```

```
  }
  debit = AH;
  BL = DL;

  printf("%s",end_msg);
  DL = BL;
  printf("%d \n",DL);
}

;console output
Andrew Robertson, x86 Lab Pt.2

Loan distributions: 2


=============================================================================================================

;No-loop trace   ;No-loop trace   ;No-loop trace   ;No-loop trace   ;No-loop trace   ;No-loop trace

-e 204
0F68:0204  FE.40
-
-e 205
0F68:0205  06.10
-
-e 206 "Andrew Robertson, x86 Lab Pt.2" 0d 0a "$"
-
-e 22F "Loan distributions: $"
-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 C7C20602      MOV     DX,0206
-t

AX=0000  BX=0000  CX=0000  DX=0206  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0104   NV UP EI PL NZ NA PO NC
0F68:0104 C6C409        MOV     AH,09
-t

AX=0900  BX=0000  CX=0000  DX=0206  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0107   NV UP EI PL NZ NA PO NC
0F68:0107 CD21          INT     21
-r ip
IP 0107
:109
-r
AX=0900  BX=0000  CX=0000  DX=0206  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0109   NV UP EI PL NZ NA PO NC
0F68:0109 B620          MOV     DH,20
-t

AX=0900  BX=0000  CX=0000  DX=2006  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=010B   NV UP EI PL NZ NA PO NC
0F68:010B 8A260402      MOV     AH,[0204]                   DS:0204=40
-t

AX=4000  BX=0000  CX=0000  DX=2006  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=010F   NV UP EI PL NZ NA PO NC
0F68:010F B200          MOV     DL,00
-t

AX=4000  BX=0000  CX=0000  DX=2000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0111   NV UP EI PL NZ NA PO NC
0F68:0111 A00502        MOV     AL,[0205]                   DS:0205=10
-t

AX=4010  BX=0000  CX=0000  DX=2000  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 28C4          SUB     AH,AL
```

```
-t

AX=3010  BX=0000  CX=0000  DX=2000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0116   NV UP EI PL NZ NA PE NC
0F68:0116 7D06        JGE     011E
-t

AX=3010  BX=0000  CX=0000  DX=2000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=011E   NV UP EI PL NZ NA PE NC
0F68:011E 88260402      MOV     [0204],AH                   DS:0204=40
-t

AX=3010  BX=0000  CX=0000  DX=2000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0122   NV UP EI PL NZ NA PE NC
0F68:0122 88D3        MOV     BL,DL
-t

AX=3010  BX=0000  CX=0000  DX=2000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0124   NV UP EI PL NZ NA PE NC
0F68:0124 BA2F02      MOV     DX,022F
-t

AX=3010  BX=0000  CX=0000  DX=022F  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0127   NV UP EI PL NZ NA PE NC
0F68:0127 B409        MOV     AH,09
-t

AX=0910  BX=0000  CX=0000  DX=022F  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0129   NV UP EI PL NZ NA PE NC
0F68:0129 CD21        INT     21
-r ip
IP 0129
:12b
-r
AX=0910  BX=0000  CX=0000  DX=022F  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=012B   NV UP EI PL NZ NA PE NC
0F68:012B B402        MOV     AH,02
-t

AX=0210  BX=0000  CX=0000  DX=022F  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=012D   NV UP EI PL NZ NA PE NC
0F68:012D 88DA        MOV     DL,BL
-t

AX=0210  BX=0000  CX=0000  DX=0200  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=012F   NV UP EI PL NZ NA PE NC
0F68:012F 80C230      ADD     DL,30
-t

AX=0210  BX=0000  CX=0000  DX=0230  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0132   NV UP EI PL NZ NA PE NC
0F68:0132 CD21        INT     21
-r ip
IP 0132
:134
-r
AX=0210  BX=0000  CX=0000  DX=0230  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0F68  ES=0F68  SS=0F68  CS=0F68  IP=0134   NV UP EI PL NZ NA PE NC
0F68:0134 CD20        INT     20
-g

Program terminated normally

;console output

-g
Andrew Robertson, x86 Lab Pt.2
Loan distributions: 0
Program terminated normally
```

FINAL CONCLUSION

This lab did its job. Since my last class in MASM was a couple years ago I felt very uneasy going in to this project and feel much more confident now. I can boil my initial gripes down to three main topics.

First, endianness. I completely forgot about big and little endian so when I entered values in memory to be copied into registers I was surprised to see the numbers were all wrong. After looking at it for a while I realized the bits were backwards then immediately the topic came back to me.

Second, conditional jumps and flags. Prior to this project I already remembered what the jump abbreviations were but I forgot how they worked. For example, I knew that JGE was jump if greater than or equal to but that was just enough o get me into trouble. When tracing the program in part 1 I couldn't figure out why JGE was evaluating true because I thought AX had to be greater than or equal to DX for it to occur. After a few traces with different values in memory I then noticed it evaluated true when AX flipped from being negative to being positive. At this point I thought it must involve the flags and after about 1 minute searching the topic via Google I was reminded that conditional jumps are triggered by flag states.

Third, interrupts. For whatever reason the idea of an interrupt still doesn't seem like something I learned before but am glad to be aware of now. INT 21 is interesting because it has the property to morph to your needs. In this program I used that property to my advantage to display both a string and a single ASCII character with the same command. After looking into it more it seems there are over a dozen different tasks INT 21 can accomplish.