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](#_Toc507167634)

[Overview 3](#_Toc507167635)

[Pre-Lab 4](#_Toc507167636)

[Problem Definition: 4](#_Toc507167637)

[Flow Chart: 4](#_Toc507167638)

[Lab Discussion 5](#_Toc507167639)

[Work Performed / Solution: 5](#_Toc507167640)

[Listing Files(s): 9](#_Toc507167641)

[Part 2 12](#_Toc507167642)

[Overview 12](#_Toc507167643)

[Pre-Lab 13](#_Toc507167644)

[Program restriction: 13](#_Toc507167645)

[Flow Chart: 14](#_Toc507167646)

[Lab Discussion 15](#_Toc507167647)

[Work Performed / Solution: 15](#_Toc507167648)

[Listing Files(s): 19](#_Toc507167649)

[Final Conclusion 31](#_Toc507167650)

# 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:



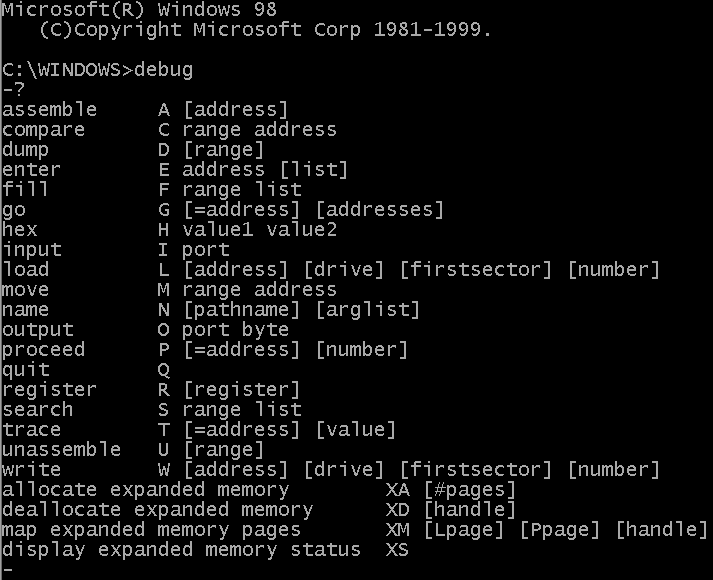
To make the flow of the program more relatable and easy to understand we were instructed to come up with a story to describe the programs actions. I described this program using a person standing in front of a vending machine trying to decide if he’s hungry enough to get something from the machine. In 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 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.

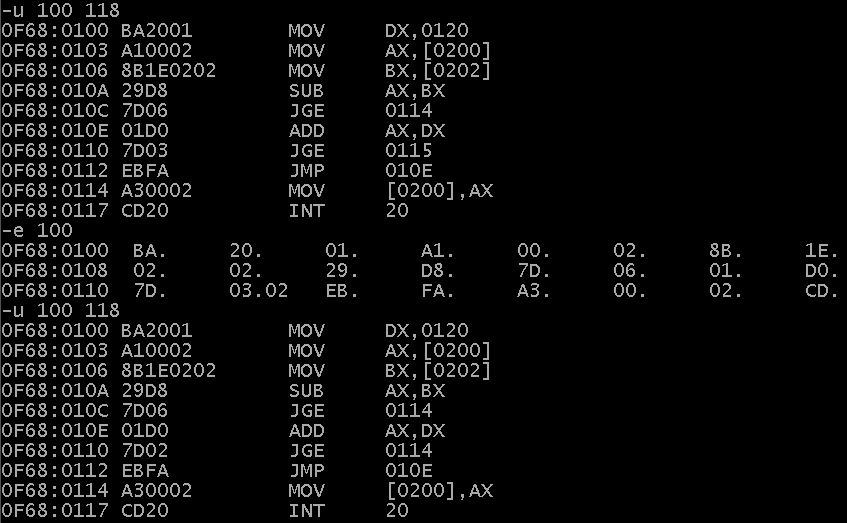
The addressing scheme is hexadecimal, displaying 0x10 or 16 bytes per line.

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.



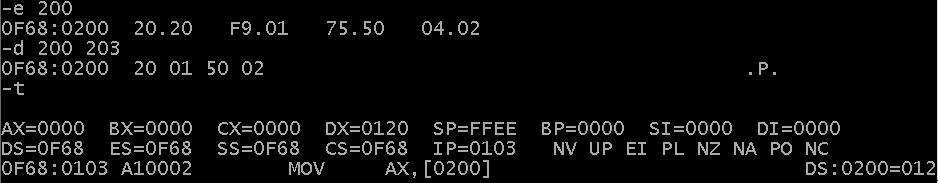
My output was not as expected the first time. I entered 7D03 for JGE 3 bytes ahead instead of 7D02 for JGE 2 bytes ahead. I found out that if space it hit without entered data when using the e command, that segment is left untouched, so I only had to enter one value to fix it instead of the whole thing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

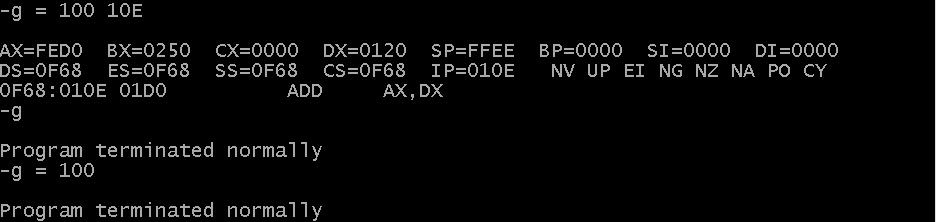
This program references memory locations for data so to have a predictable running program we must put data there we want to use. If these locations were just to store data, then we wouldn’t need to do this.



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.



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):

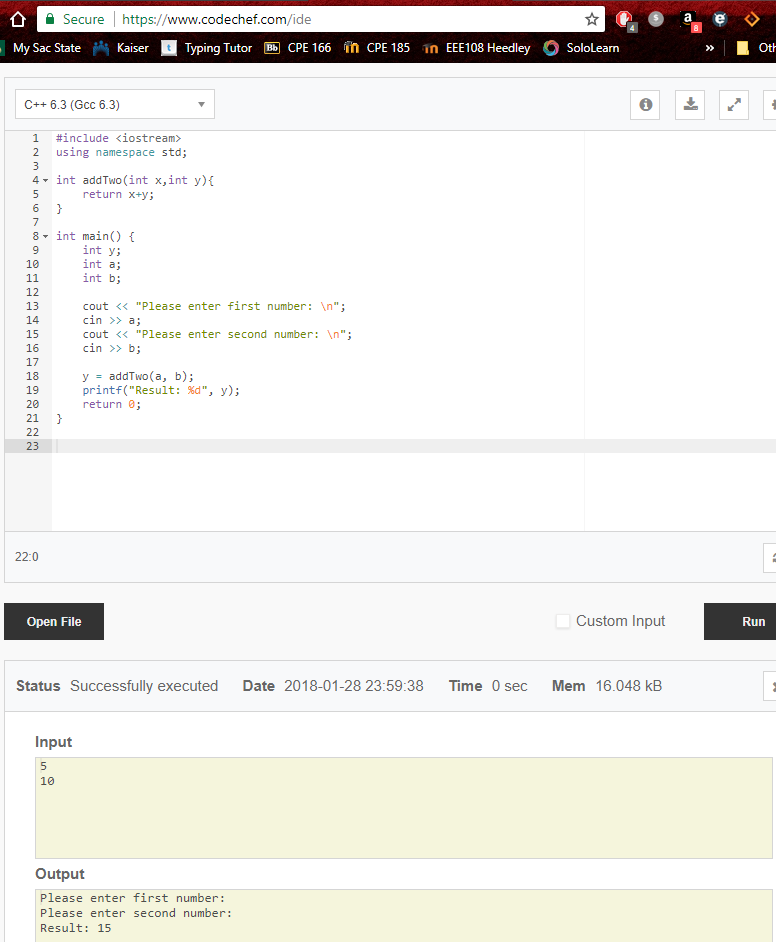
Tracing chart



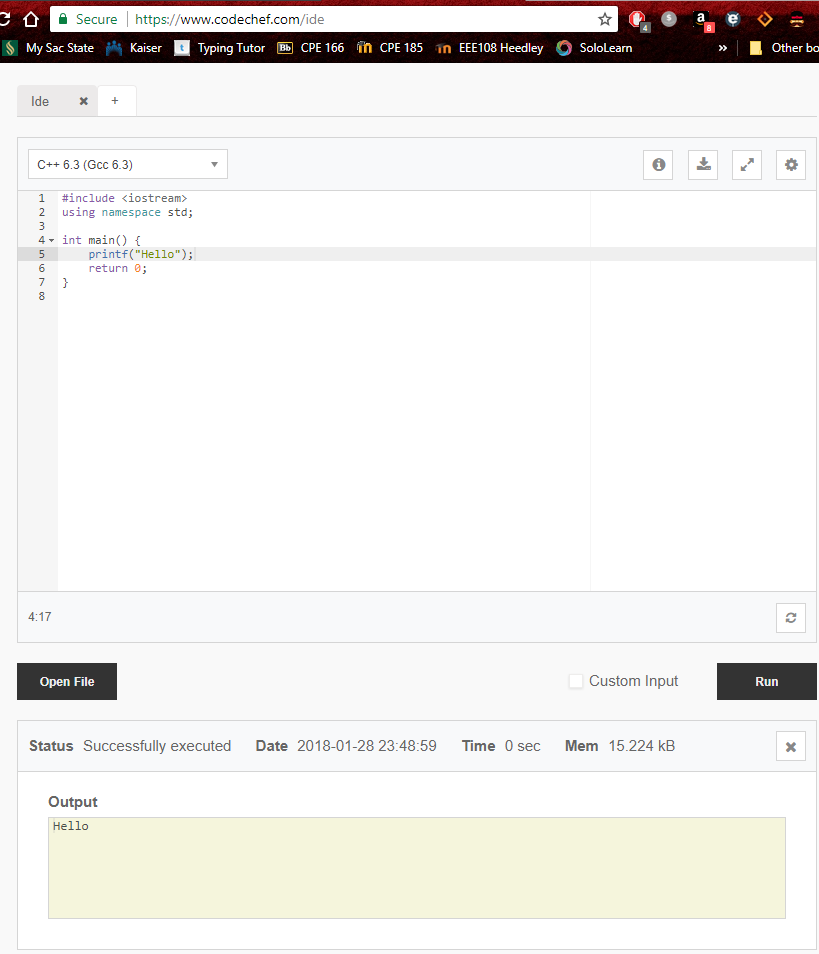


Hand assembly

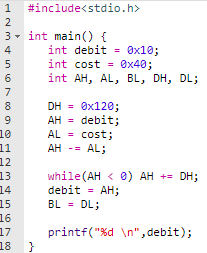
C program to add two numbers



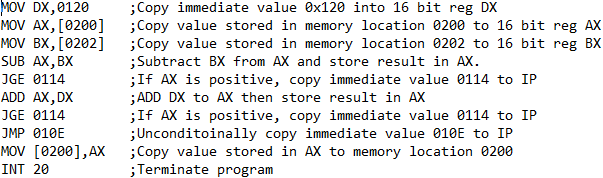
C program to output Hello to standard out



C program to mimic the MASM program



Commented MASM program



# Part 2

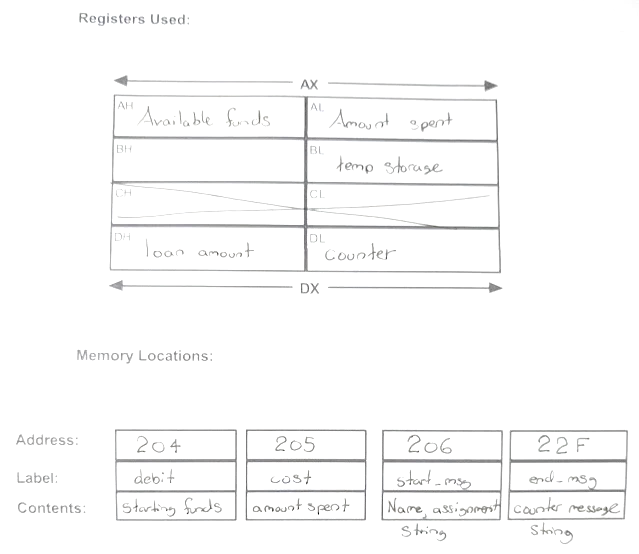
## Overview

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.



### 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.



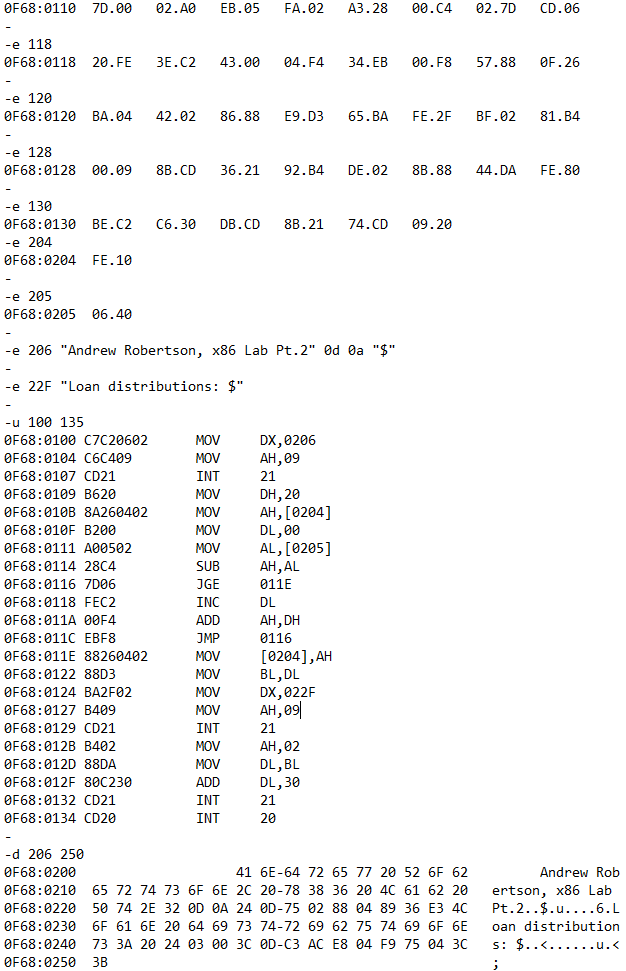
## Lab Discussion

### 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:



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):













;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.