Daniel Komac

CPE 185: Introduction to Microprocessors

Lab Session: Wednesday

**Lab 1: x86**

Instructor: Chris Moyer

Date: November 17th, 2015

**Introduction:**

The purpose of this lab was to introduce us to the Intel x86 architecture. This lab consisted of 3 parts using DOS in a windows 98VM. The virtual machine was needed to access MASM as well, using DOS allowed us to debug in assembly language and work on hardware level.

**Part 1: Debug and C refresher:**

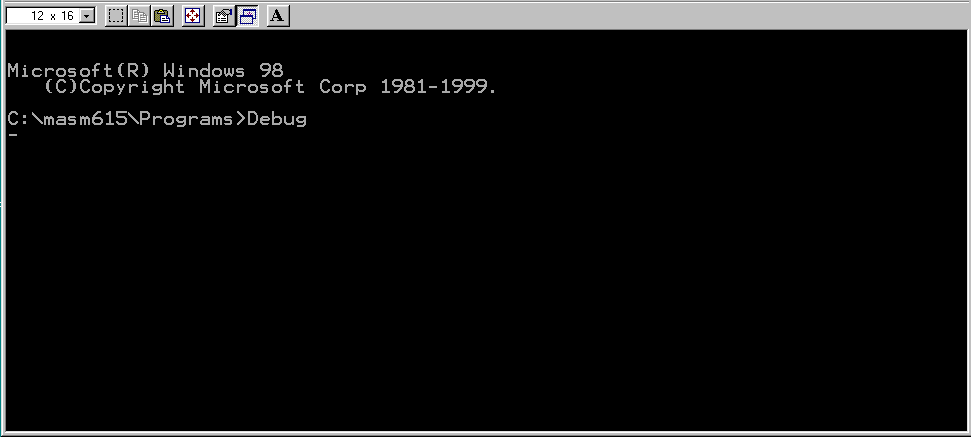
**Purpose:**

Understanding how to use DOS Debug and getting us re-familiarized with a programming language that is very important in programming many of the boards that we work with such as Arduino or Propeller.

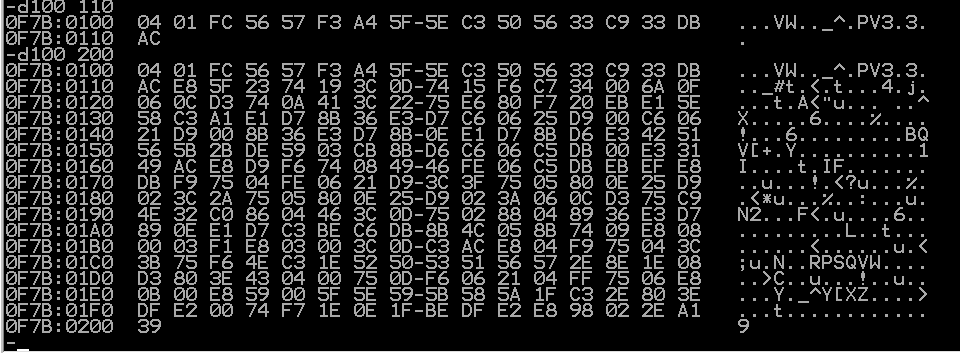
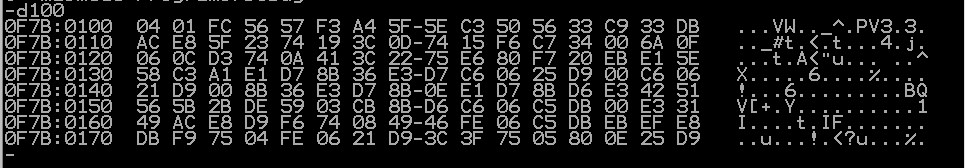
**Analysis:**

Within the DOS window we are able to use DOS commands that show us useful data. But in order to learn the commands first we use the help command which is used in DOS’s debug mode

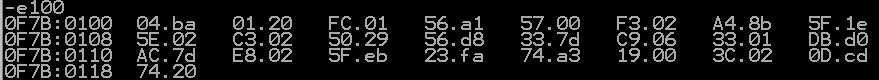
First we enter Debug:



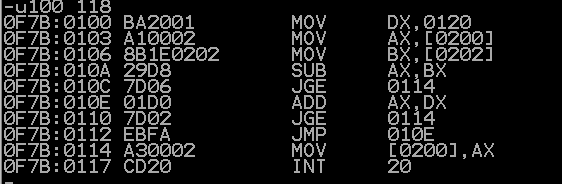
Then the dump command to see what is in memory at the moment.



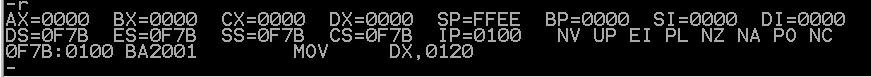
After finding out what we have in our memory we edit the memory segment of 0100 to 0118 with the e(enter) command



After we entered our data we were able to unassembled the program and see how it works and what exact directions we have given the computer after we had entered data into memory location 100. This is the result



After that we check the value of our registers to see if we are in the right place



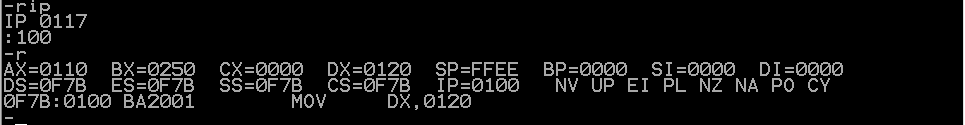
All our registers are clear and we are on the first line of code, so now we will trace the program with the t command.



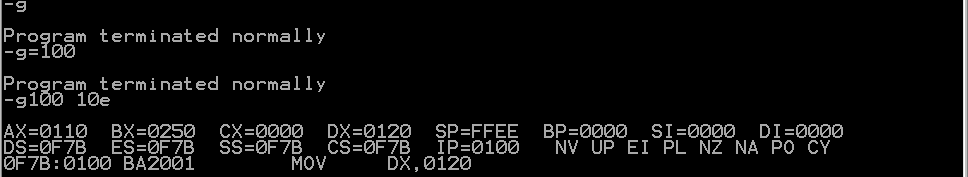
I executed this program twice with different values. These were my results for tracing



After doing the trace commands we can re position or code segment by using the rip command



We can achieve the same result out of the go command, or g



The go command executes the code from the beginning source until it sees an int command which terminates the program or if there is a declaration for a line termination. For example the g100 t 10e goes to code segment 10e then stops. The int command terminates the program, the INT command found in the program is INT 20, this one is specific for terminating programs this is why we get the message at the end of the run.

**C Programs:**

These were basic C programs to refresh our memory on C language and it was fairly simple by making us do simple output, addition, and implementation of the assembly code in terms of C language to understand how it processes.

**Code:**

#include <stdio.h>

#include <iostream>

void hello();

int addNumbers(int x, int y);

int x86(int x, int y, int z);

void main(int argc, char \*argv[])

{

//Variables

int f; int s; int t;

//hello World program

hello();//Calls hello function

//Adding Numbers program

printf("\n Adding numbers: \n");//Setting the numbers for being added

printf("Choose your first number: ");//Asking user input for first number

std::cin >> f;

printf("\nChoose your second number: ");//Asking user input for second number

std::cin >> s;

std::cout <<"\nYour number equals: " << addNumbers(f,s);//Displays final result by calling addNumber function

getchar();//Delay for display

getchar();

//x86 moc Program

printf("\n Enter over-draft fee: ");//Asks user for over-draft from savings

std::cin >> s;

printf("\n Enter bank account total: ");//Asks user for beginning bank acount number

std::cin >> f;

printf("\n Enter withdrawal amount: ");//Asks user how much is being withdrawn

std::cin >> t;

int storage;//Initialzes a holding variable for the bank account

storage = x86(s,f,t);//Calls x86 moc program

std::cout<<"\nYour bank account after overdrafts is: "<< storage;//Displays final bank total

getchar();//Delay for displaying

getchar();

}

void hello()

{

printf("\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n");

printf("\n Hello World \n");//Display hello world

printf("\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n");

}

int addNumbers(int x, int y)

{

return x+y;//Add both parameters

}

int x86(int x, int y, int z)

{

int overFee = x;//Overdraft fee is placed here

int bankAcc = y;//Beginning bank account starts here

int withdrawal = z;//Amount to pull out from bank account

bankAcc -= withdrawal;//Take away the amount desired from bank account

while(bankAcc<=0)//Make sure the amount being taken out doesn't leave

{ //the account negative.

bankAcc += overFee;//Increment the bank account by pulling x amount from

} //savings

return bankAcc;//Return the bank account after withdrawl and overdrafting

}

I decided to make methods that do the work for me and combo the labs into one source file as to avoid having multiple codes to switch between and run. And to show the logic of the C program I will provide a flowchart:



**Part II: Hand Assembly**

Before starting this section I had to think over how I would alter the code to exclude the C register and to use memory location 0464. This was fairly simple for excluding registers because the code initially does not use the C register I just had to modify the code to 8 bit and change the memory addresses used.

Modifying to 8 bit was relative simple since the code hardly changed just the memory we entered into code segment 100. The code is altered and changed the code to a different form, the hand assembly flow chart shows exactly how I did this. The following assembly chart shows the step by step process:



Now if I was to implement a counter into the code I would have to change a couple lines of code to store a value within a register, the register I will use for my counter will be the BH register. This register will hold a value that is how many times the overdraft loop executes, and then it will display it at the end of the program.

-u100 138

0F6C:0100 B025 MOV DL,42

0F6C:0102 8A1E8605 MOV BL,[0465]

0F6C:0106 8A168805 MOV AL,[0464]

0F6C:010A 2ADA SUB AL,BL

0F6C:010C B700 MOV BH,00

0F6C:010E 883E0005 MOV [0466],BH

0F6C:0112 7D0A JGE 011E

0F6C:0114 FEC7 INC BH

0F6C:0116 883E0005 MOV [0466],BH

0F6C:011A 02D8 ADD BL,AL

0F6C:011C EBF0 JMP 010E

0F6C:011E A28605 MOV [0464],AL

0F6C:0121 8A3E0005 MOV BH,[0466]

0F6C:0125 80C730 ADD BH,30

0F6C:0128 883E0005 MOV [0466],BH

0F6C:012C BA0005 MOV DX,0500

0F6C:012F B409 MOV AH,09

0F6C:0131 CD21 INT 21

0F6C:0133 BA0002 MOV DX,0200

0F6C:0136 CD21 INT 21

0F6C:0138 CD20 INT 20

-g=100

2This class is awesome.

Program terminated normally

I used the memory location 466 and the ‘$’ to print whole strings and I also add 30 to BH so that the number being printed is the ASCII value.

**Part III:**

Unfortunately I was unable to get to the MASM portion of this lab and therefore am unable to write about it.