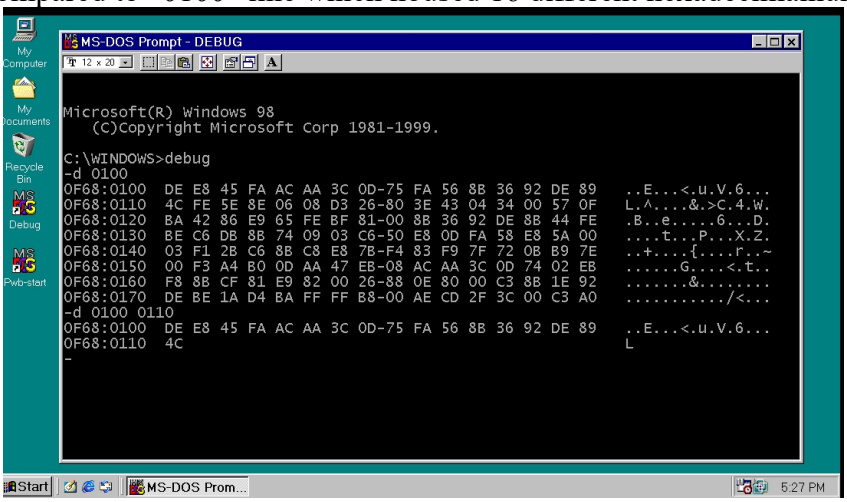
Andrew Stites
EEE 174 – CpE185 Section 2
Summer Session 2
Lab 1
Sean Kennedy

Introduction:

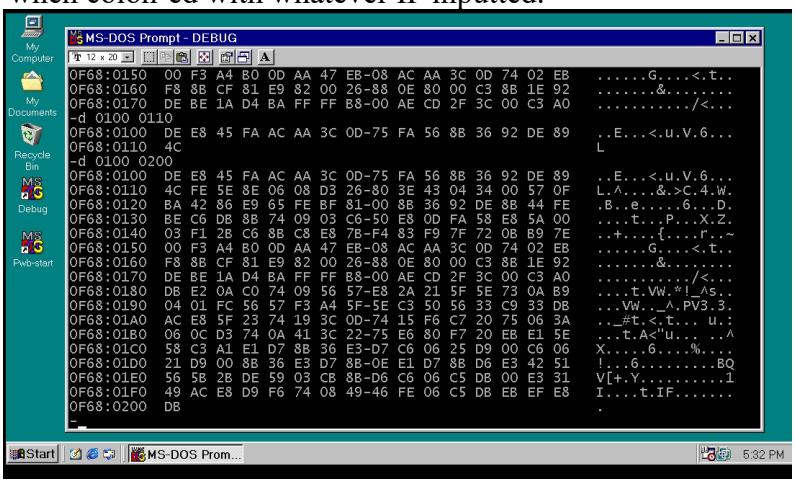
This lab allowed familiarity with assembling programs by implementing and tweaking machine code in hexadecimal. It was also a refresher for those whom had not done any coding in a long time and an introduction to MS-DOS working in a virtual machine. In MS-DOS, we learned the basics of debugging a program utilizing six different commands. This was the basics for microprocessors and utilizing the software and learning techniques that will be implemented in future endeavors regarding hardware or other software. In addition, we utilized spreadsheets to portray our programs tracing pattern in regards to memory location and value storage. We also used spreadsheets to manually hand assemble our program utilizing binary to portray hexadecimal values in Little Endian fashion.

Part One:

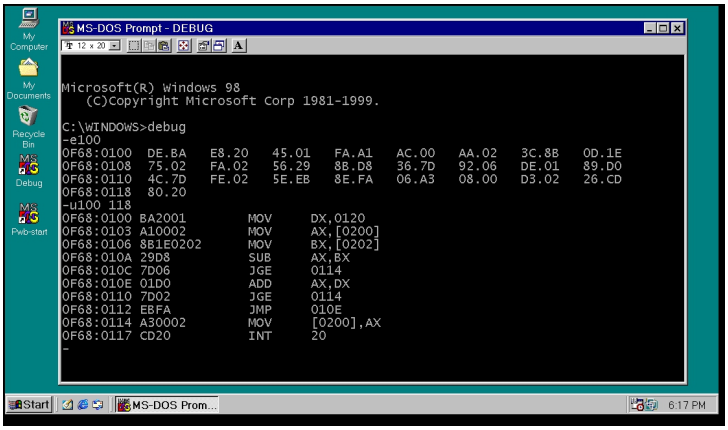
After installing VMware workstation 16 onto the computer, I familiarized myself by opening the start menu and locating MS-DOS. I typed “debug” after the “C:/WIndows>” prompt line then added “?” To the line that followed to receive a list of different commands. I exited out and logged back in with debug, but used “d” for “dump” while adding “0100” to the end. I hit enter and a series of hexadecimal numbers appeared in the middle while the left side housed what seemed like addresses starting at “0100” leading up to “0170” incrementing by 10. The right side of the command prompt displayed a series of random characters sporadically placed. Next, I typed “d 0100 0110” and only two addresses appeared with their respective lines of hexadecimal numbers: “0100” and “0110”. “0110”’s line only had one hexadecimal number: “4C” compared to “0100” line which housed 16 different hexadecimalnumbers.



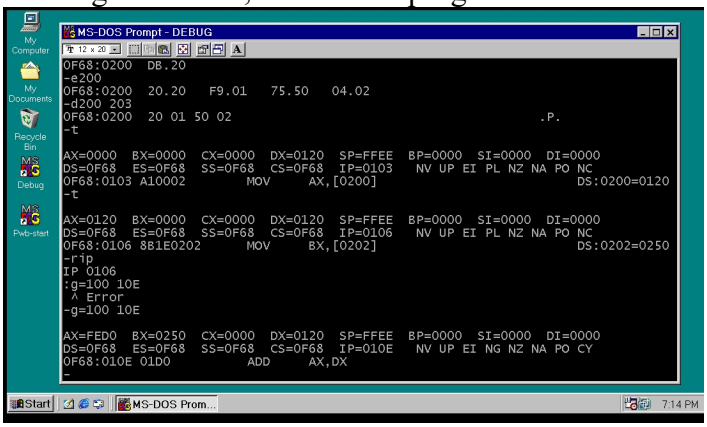
I finished the test by inputting “d 0100 0200” and it resulted in the addresses incrementing by 10 till they hit “0200”. The dump command “d” displays all the addresses desired utilizing hexadecimal as a factor of value storage. The “CS” or code segment was rigid at “0F68” when colon’ed with whatever IP inputted.



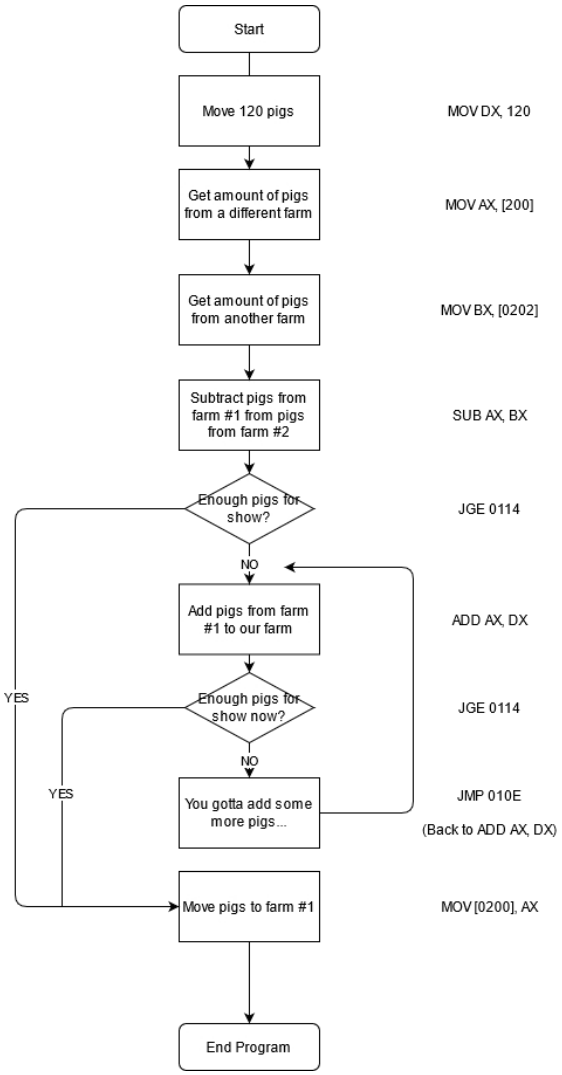
I entered “debug” then “e” for “enter” and started at memory location “0100”. I entered the respective machine code for each, pressing space to move from one another. I finished with the “0118” address with having only one machine code inputted.I utilized the “unassemble” command “u” to print out the program listing. When comparing my program listing to the one issued, I found no flaws and both were identical disregarding the differing CS. The machine code translates to the mnemonics, registers, and number characters in conjunction with the addresses. For instance, the machine language “BA2001” at address “0100” converts to the mnemonic “MOV” for machine code “B” and “A” for register “DX”. The “2001” is in Little Endian fashion so the most significant bit is the “01” and the least is the “20”. This gets flipped to Big Endian fashion for the addresses “101” and “102”.



I entered the “register modify” command “r” to set the IP (instruction pointer) to code segment (CS) “0100”, which is the beginning of the program. The result printed matched what was issued disregarding all the registers that had my differing CS value of 0F68. Before I traced through the program with “trace” command “t”, I utilized the “enter” command “e” at address “0200” through “0203” to apply the values of “0120” and “0250” using machine language. After properly entering the values at the addresses “0200” through “0203” for the registers “AX” and “BX”, I used the “dump” command “d” to display the values that were inputted into the addresses in Little Endian fashion. I officially used the “trace” command “t” to go through the program by displaying the registers with their respective values. I used “rip” to access the IP’s registered address and changed it to “0100” by utilizing the “go” command “g” to set the starting point to “0100” and have a break point at the address “10E” which correlated to the mnemonic “ADD” and registers “AX, DX” in the program.



Flow Chart for Part One:

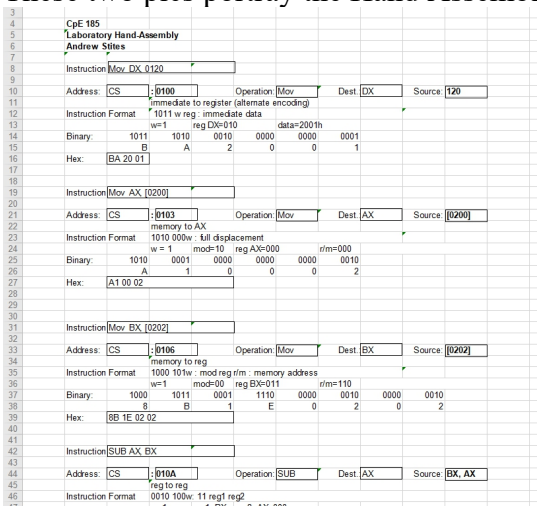


Commented Code:

MOV DX,0120 // move hex value 0120 into register DX
MOV AX,[0200] // move 2 bytes from memory location 0200 into reg AX
MOV BX,[0202] // move 2 bytes from memory location 0200 into reg BX
SUB AX,BX // subtract reg BX from AX and store in AX
JGE 0114 // jumps to memory location 114 if the result is >0
ADD AX,DX // add reg DX to AX and store in AX
JGE 0114 // jumps to memory location 0114 if the result is >0
JMP 010E // jump unconditionally to location 010E
MOV [0200],AX // move the contents of AX to memory location 0200
INT 20 // BIOS service interrupt 20; ends the program

Supporting Pics:

These two pics portray the Hand Assembly and the Tracing Chart:



CpE 185
Laboratory Hand-Assembly
Andrew Stites

Instruction: Mov DX, 0120

Address: CS : 0100 Operation: Mov Dest: DX Source: 120
immediate to register (alternate encoding)
Instruction Format: 1011 w reg : immediate data
w=1 reg DX=010 data=2001h
Binary: 1011 1010 0010 0000 0000 0001
B A 2 0 0 1
Hex: BA 20 01

Instruction: Mov AX, [0200]

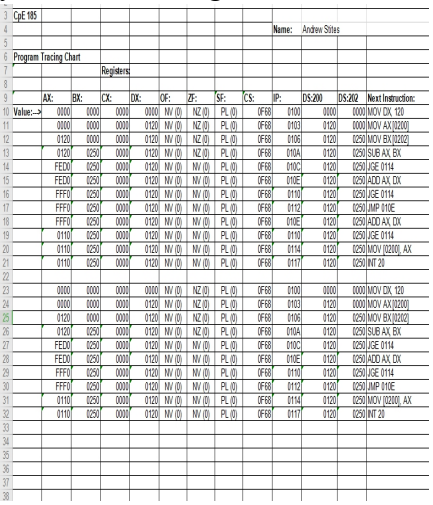
Address: CS : 0103 Operation: Mov Dest: AX Source: [0200]
memory to AX
Instruction Format: 1010 000w : full displacement
w = 1 mod=10 reg AX=000 r/m=000
Binary: 1010 0001 0000 0000 0000 0010
A 1 0 0 0 2
Hex: A1 00 02

Instruction: Mov BX, [0202]

Address: CS : 0106 Operation: Mov Dest: BX Source: [0202]
memory to reg
Instruction Format: 1000 101w : mod reg r/m : memory address
w=1 mod=00 reg BX=011 r/m=110
Binary: 1000 1011 0001 1110 0000 0010 0000 0010
8 B 1 E 0 2 0 2
Hex: 8B 1E 02 02

Instruction: SUB AX, BX

Address: CS : 010A Operation: SUB Dest: AX Source: BX, AX
reg to reg
Instruction Format: 0010 100w: 11 reg1 reg2



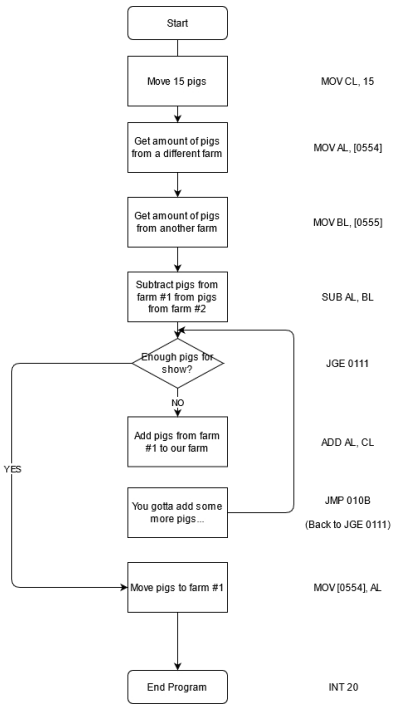
CpE 185
Name: Andrew Stites
Program Tracing Chart
Registers:

	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	0000	0000	MOV DX, 120
	0000	0000	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	0103	0120	0000	MOV AX,[0200]
	0120	0000	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	0106	0120	0250	MOV BX,[0202]
	0120	0250	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	010A	0120	0250	SUB AX, BX
	FED0	0250	0000	0120	NV (0)	NV (0)	PL (0)	0F68	010C	0120	0250	JGE 0114
	FED0	0250	0000	0120	NV (0)	NV (0)	PL (0)	0F68	010E	0120	0250	ADD AX, DX
	FFF0	0250	0000	0120	NV (0)	NV (0)	PL (0)	0F68	0110	0120	0250	JGE 0114
	FFF0	0250	0000	0120	NV (0)	NV (0)	PL (0)	0F68	0112	0120	0250	JMP 010E
	FFF0	0250	0000	0120	NV (0)	NV (0)	PL (0)	0F68	010E	0120	0250	ADD AX, DX
	0110	0250	0000	0120	NV (0)	NV (0)	PL (0)	0F68	0110	0120	0250	JGE 0114
	0110	0250	0000	0120	NV (0)	NV (0)	PL (0)	0F68	0114	0120	0250	MOV [0200], AX
	0110	0250	0000	0120	NV (0)	NV (0)	PL (0)	0F68	0117	0120	0250	INT 20
	0000	0000	0000	0000	NV (0)	NZ (0)	PL (0)	0F68	0100	0000	0000	MOV DX, 120
	0000	0000	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	0103	0120	0000	MOV AX,[0200]
	0120	0000	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	0106	0120	0250	MOV BX,[0202]
	0120	0250	0000	0120	NV (0)	NZ (0)	PL (0)	0F68	010A	0120	0250	SUB AX, BX
	FED0	0250	0000	0120	NV (0)	NV (0)	PL (0)	0F68	010C	0120	0250	JGE 0114
	FED0	0250	0000	0120	NV (0)	NV (0)	PL (0)	0F68	010E	0120	0250	ADD AX, DX
	FFF0	0250	0000	0120	NV (0)	NV (0)	PL (0)	0F68	0110	0120	0250	JGE 0114
	FFF0	0250	0000	0120	NV (0)	NV (0)	PL (0)	0F68	0112	0120	0250	JMP 010E
	0110	0250	0000	0120	NV (0)	NV (0)	PL (0)	0F68	0114	0120	0250	MOV [0200], AX
	0110	0250	0000	0120	NV (0)	NV (0)	PL (0)	0F68	0117	0120	0250	INT 20

Part Two:

For part two, we utilized the same code from part one, but converted it from 16-bit to 8 bit. This was done by changing the registers from their full 16-bit version, i.e. “AX”, “BX”, etc. to their 8 bit forms: “AL”, “BL”, etc.. There was a stipulation placed on coding part two entailing only using a certain memory location based on one’s first initial of one’s first name and being barred from utilizing a certain register depending on one’s first initial of one’s last name. My memory location chosen was “0554” in Little Endian notation and I was stripped of any register starting with“D”. However, since “DX” is necessary for message displaying, I was allowed to use it for such.

Flow Chart for Part Two:



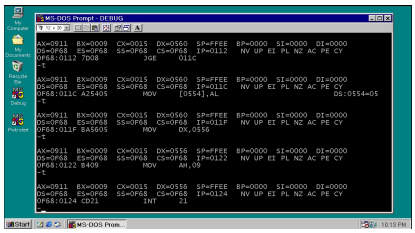
Commented 8-bit Code:

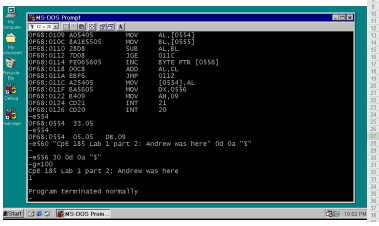
/*Since my last name starts with an ‘S’, I could not use the D register. Since my first name started with ‘A’, my memory location is [0554]*/
MOV CL, 15 // move value 15 into register DX
MOV AL, [0554] // move 1 byte from memory location 0554 into reg AL
MOV BL, [0555] // move 1 byte from memory location 0555 into reg BL
SUB AL,BL // subtract reg BL from AL and store in AL
JGE 0111 // jumps to memory location 0111 if the result is >0
ADD AL, CL // add reg CL to AL and store in AL
JMP 10B // jump unconditionally to location 010B
MOV [0554], AL // move the contents of AL to memory location 0554
INT 20 // BIOS service interrupt 20; ends the program

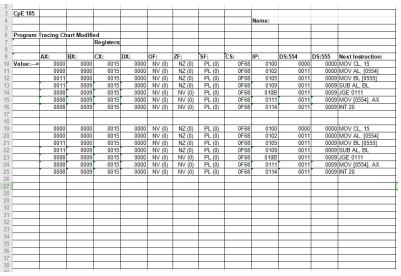
Commented 8-bit Code with Message Code:

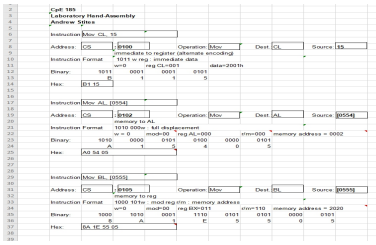
/*Since my last name starts with an ‘S’, I could not use the D register. Since my first name started with ‘A’, my memory location is [0554]*/
MOV DX, 0560// DX must be used to display strings due to the address needed
MOV AH, 09// 09 sets the BIOS service to display the message
INT 21 //DOS interrupt to display message
MOV CL, 15 // move value 15 into register DX
MOV AL, [0554] // move 1 byte from memory location 0554 into reg AL
MOV BL, [0555] // move 1 byte from memory location 0555 into reg BL
SUB AL,BL // subtract reg BL from AL and store in AL
JGE 011C // jumps to memory location 0111 if the result is >0
INC BYTE PTR [0556]//increments a counter for every iteration of the loop
ADD AL, CL // add reg CL to AL and store in AL
JMP 112 // jump unconditionally to location 010B
MOV [0554], AL // move the contents of AL to memory location 0554
MOV DX, 0560// DX must be used to display strings due to the address needed
MOV AH, 09// 09 sets the BIOS service to display the message
INT 21 //DOS interrupt to display message
INT 20 // BIOS service interrupt 20; ends the program

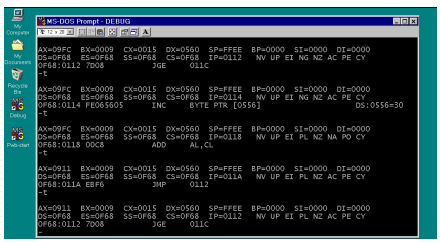
Supporting Pics:

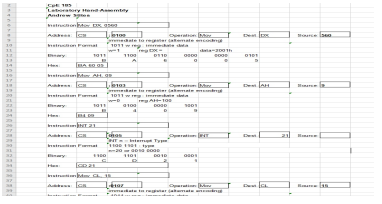












Conclusion:

When compiling all the information I have learned from CSC 35, CSC 60, and CpE 64, it was refreshing to see it all being applied in a real-life application. The newest feature that I learned was the implementation of machine code in conjunction with assembly. Utilizing “debug” and MS-DOS was a great learning experience due to having easily digestible steps. The instructions were recognized due to CSC 35’s teachings. MOV is used to move data either from an immediate to a register or register to register; SUB takes the difference from register 1 with register 2, or an immediate, and puts that new value into register 1. ADD creates the sum of the two registers or register and immediate and stores it into register 1. JGE is an acronym for “jump if greater or equal” and was used to jump to a specific address in the code if the value given was greater than “0”. INT 20h is the BIOS service interrupt to end the program safely and securely. The debug commands consist of six that we used: command “d”, or “dump”, was used to display all the addresses desired utilizing hexadecimal as a factor of value storage. The command “e”, or “enter”, allows for inputting hexadecimal values in the respective instruction pointers. The command “u”, or “unassemble”, prints out the program listing after the programmer has inputted the correct machine code using “enter”. The command “r”, or “register modify”, to set the code to a certain IP(instruction pointer), for instance, setting the IP to “0100” for the start of the program. The command “g”, or “go”, is a versatile command allowing for the running of the program. It can be set at a certain memory location by inputting “g=100” and can have a breakpoint allowing for the program to run to a certain point and stop. The command “t”, or “trace”, allows for the programmer to go through the program step-by-step for visuals into how the values and addresses are functioning.