Hand-Assembly, Using DEBUG

Introduction:
In this Hand-Assembly Lab, you will develop an 8-bit version of the program from Debug Introduction Lab, using byte-size data and registers (Debug Introduction Lab uses word size, 16 bit registers and data). Your program must meet the following specifications:

1. Use the JGE instruction only once (only one conditional jump).
2. Make no use of register (obtain from your Lab Instructor) _____, neither its low nor high byte.
3. Use consecutive memory locations for data starting at the address _____; you will obtain from your Lab Instructor.

Note: Each person will have a unique register to not use and an address for data. Also DO NOT use the Assembler in DEBUG. Refer to the hand-assembly examples.

4. After you have your program hand assembled and running, using the assemble command modify your program to display your name and title of your program (see the "Welcome to EEE174" program).
5. Modify your program to keep track of how many times the overdraft (or bail out) has been added to the account.

Pre-Lab Work:
1. Identify the registers and memory locations you want to use on the attached Register Template. Use descriptive words such as “old balance”, “debit”, “over-draft or bailout” to describe what numeric values are and how they relate to the program operation.
2. Create a Flow Chart for your program (see handout on Flow Charting). Your instructor will want to see your flow chart at the beginning of the lab, make sure you instructor signs you off before you proceed with entering your program.
3. Hand-assemble the instructions using the hand-assembly template (see attached), you may need to use more than one page. Your instructor will want to see your Hand-assembly work.

Laboratory Work:
1. Use the DOS DEBUG command “e” to load the results of your hand-assembly work into memory starting at location CS: 0100 (use the “CS”, Code Segment, value given by your computer). Use the DOS DEBUG command “u” to un-assemble your program to verify it was loaded correctly and is executing the instructions you programmed. Go back to your hand-assembly template to fix errors that you find.
2. When your program is working correctly, copy your program listing into a text editor and add commented code (a written comment on what the instruction does in context to the entire program), behind each machine language instruction. Use the tracing chart from the Debug Introduction Lab, and trace two runs of your program (at least one trace with the variables set to cause the program to loop more than once).
3. Demonstrate your program to the Lab Instructor. Your Lab instructor will want to see:
   1. Your updated pre-lab Flow Chart showing the machine instructions for each symbol on the chart
   2. Your Hand-assembly work
   3. Commented Code
   4. Be able to discuss the program (you can be asked about any instruction and you should be able to discuss that instruction)

IMPORTANT NOTE:
Throughout the semester, laboratory reports must be submitted to the laboratory instructor during the laboratory period in which they are due. Lab reports that are one week late will receive ½ credit. Reports later than two weeks will receive no credit.

Lab Report Hand Assembly Lab Due: Week 6
### Registers Used:

<table>
<thead>
<tr>
<th>AH</th>
<th>AL</th>
</tr>
</thead>
<tbody>
<tr>
<td>BH</td>
<td>BL</td>
</tr>
<tr>
<td>CH</td>
<td>CL</td>
</tr>
<tr>
<td>DH</td>
<td>DL</td>
</tr>
</tbody>
</table>

### Memory Locations:

<table>
<thead>
<tr>
<th>Address:</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Label:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contents:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


This is an example Hello program (in x86 code)
for use in DOS Debug
Written by Dennis Dahlquist 1/30/02
---------------------------------------------------------------;
 debug
 A 100
 Mov DX, 200 ; load DX with the value of location of the message
 mov AH, 09 ; set the BIOS service to display the message
 int 21 ; DOS interrupt to display message
 int 20 ; terminate program and return to DOS
---------------------------------------------------------------;
E 200 "Hello, Welcome to EEE174" 0d 0a "$"
 ; Message Data, 0d & 0a - cr and lf, $ - end of string
Results:
-u100 108
290D:0100 BA0002 MOV DX,0200
290D:0103 B409 MOV AH,09
290D:0105 CD21 INT 21
290D:0107 CD20 INT 20
-
-d200 21a
290D:0200 48 65 6C 6C 6F 2C 20 57-65 6C 63 6F 6D 65 20 74   Hello, Welcome to
290D:0210 6F 20 45 45 45 31 37 34-0D 0A 24                  o EEE174..$
-
-g=100
Hello, Welcome to EEE174
Program terminated normally
-
C:\WINDOWS>debug
-?
assemble A [address]
compare C range address
dump D [range]
enter E address [list]
fill F range list
go G [=address] [addresses]
hex H value1 value2
input I port
load L [address] [drive] [firstsector] [number]
move M range address
name N [pathname] [arglist]
output O port byte
proceed P [=address] [number]
quit Q
register R [register]
search S range list
trace T [=address] [value]
unassemble U [range]
write W [address] [drive] [firstsector] [number]
allocate expanded memory XA [#pages]
deallocate expanded memory XD [handle]
map expanded memory pages XM [Lpage] [Ppage] [handle]
display expanded memory status XS
-
Instruction:

Address: CS __ : __  Operation: __  Dest: __  Source: __

Instruction Format

Binary:

Hex: __

Instruction:

Address: CS __ : __  Operation: __  Dest: __  Source: __

Instruction Format

Binary:

Hex: __

Instruction:

Address: CS __ : __  Operation: __  Dest: __  Source: __

Instruction Format

Binary:

Hex: __

Instruction:

Address: CS __ : __  Operation: __  Dest: __  Source: __

Instruction Format

Binary:

Hex: __

Instruction:

Address: CS __ : __  Operation: __  Dest: __  Source: __

Instruction Format

Binary:

Hex: __
Example

Instruction: **Mov DL, 25**

Address: CS: 0100

Operation: Mov

Dest.: DL

Source: 25

Instruction Format:

- Immediate to register (alternate encoding)
- w=0
- reg DL=010
- data=0010 0101

Binary: 1011 0010 0010 0101

Hex: B2 25

Instruction: **Mov BL, [0200]**

Address: CS: 0102

Operation: Mov

Dest.: BL

Source: [0200]

Instruction Format:

- Memory to reg
- mod=00
- reg BL=011
- r/m=110

Binary: 1000 1010 0001 1110 0000 0000 0000 0010

Hex: 8A 1E 00 02

Instruction: **ADD BL, DL**

Address: CS: 0106

Operation: ADD

Dest.: BL

Source: DL, BL

Instruction Format:

- Register2 to register1
- w=0
- reg2=DL=010
- reg1=BL=011

Binary: 0000 0010 1101 1010

Hex: 02 DA

Instruction: **INT 20**

Address: CS: 0108

Operation: INT

Dest.: 20

Source: 

Instruction Format:

- INT n – Interrupt Type
- n=20 or 0010 0000

Binary: 1100 1101 0010 0000

Hex: CD 20
Instruction: JMP 110
Address: CS: 0114 Operation: JMP Dest: 110 Source: 
JMP – Unconditional Jump (to same segment)
Instruction Format: short 1110 1011: 8-bit displacement
Binary: 1110 1011 1111 1010
Hex: EB FA

Instruction: JGE 11A
Address: CS: 112 Operation: Dest: Source: 
Jcc – Jump if Condition is Met
Instruction Format: 8-bit displacement 0111 tttn: 8-bit displacement JGE => tttn = 1101 displacement = 06 or
Binary: 0111 1101 0000 0110
Hex: 7D 06
<table>
<thead>
<tr>
<th>AX</th>
<th>BX</th>
<th>CX</th>
<th>DX</th>
<th>OF</th>
<th>ZF</th>
<th>SF</th>
<th>CS</th>
<th>IP</th>
<th>DS:200</th>
<th>DS:202</th>
<th>Next Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>