COS 335 Spring 2010
Assignment 5 Due Thursday March 17

For an introduction to assembler, we will run some simple series of instructions in DEBUG (or GRDB, as you prefer). These need to be done on a Windows machine with a 32-bit OS (any version of Windows will do). For Linux you can try DOSBOX, but it's pretty clunky. For 64-bit Windows OS, try a different machine (Sorry!). All future work can be done on 32 or 64 bit Windows or Linux.

Note: if you terminate debug programs with the INT 20 instruction, execution will always terminate when INT 20 is executed, and registers will be reset.

1. (1 pt) Write a program in DEBUG that does the four additions in HW1 Question 5: "Assuming an 8-bit representation, list 4 pairs of bit strings whose binary interpretations as unsigned and 2's complement numbers have the following properties: a) Both signed and unsigned addition is correct; b) Signed addition is correct, but unsigned is not; c) Unsigned addition is correct, but signed is not; d) Neither signed nor unsigned addition is correct." Since incorrect unsigned addition is represented by the Carry Flag set (CY) and incorrect signed addition is represented by the Overflow flag set (OV) you will be able to see these flags as you trace the program in Debug. Trace the program and submit a screen dump of the Debug trace showing the flags in each of the four cases, as well as a screen dump showing your program with the U command.
Example: correct for signed and unsigned interpretations:

```
mov al, 1
add al, 1
```

2. (1 pt) Write a short program consisting of a single MOV instruction followed by a series of 4 ADD instructions that will set each of the flags in this order: OF, CF, ZF, SF. In other words the first ADD should set OF and you don't have to worry about CF, SF, ZF. The second ADD should set CF, etc. Trace the program and submit a screen dump of the Debug trace showing the flags in each of the four cases, as well as a screen dump showing your program with the U command.

3. (1 pt) Assemble the following program that gets the system time:

```
mov ah, 2C
int 21
```

Use the "P" (Proceed) command when you reach INT 21. Then use R to display registers. The current time is returned as CL = hours; CL = minutes, DH = seconds, and DL = hundredths.

Submit a screen dump showing the registers and then decode the values into human readable form.

4. (1 pt) Repeat number 3 to display the current date rather than the current time, using function 2Ah (Get Date).
Function 2A Returns:

```
AL Day of Week (Sunday = 0)     CX  Year (1980-2099)
DH Month (1-12)                 DL Day (1-31)
```

Note that the year is a 16-bit integer, unlike the month and day, where 0 is 1980.
Submit a screen dump showing the registers and then decode the values into human readable form.

5. (1 pt) Use the floating point processor to store the value of PI to memory

```
FLDPI                ; load PI into ST(0)
FSTP DWORD [200]     ; store and pop into [200] as a 32-bit float
INT 20
```

After running the program, use the command D200 to display memory, and decode enough of the float at address 200 to show that it is indeed pi to 3 decimal digits (3.141...) – just zero out and ignore some of the least significant bits.

6. (1 pt) Compute the square root of 2. We'll use the FLD1 (Load constant 1) to compute 1+1 = 2 first. Every time a number is loaded into the 8087, it is placed into the stack top ST(0), and other values are pushed lower in the stack. The FADD instructtion with no operands computes ST(1) = ST(0) + ST(1), leaving ST(0) unaltered. But we can FADDP (Floating ADD and Pop), which computes ST(1) = ST(0) + ST(1), then pops the stack, leaving the former ST(1) now inST(0). The square root instruction (FSQRT) computes the square root of ST(0) and leaves the result in ST(0)

```
                                ST(0)   ST(1)
FLD1                              1      --
FLD1                              1       1
FADDP                             2      --
FSQRT                          1.414     --
FSTP DWORD [200]                  --     --
INT 20
```

Then D 200 as in (5) and decode the result. Submit the screen dump of memory, your decoding of the result (3 decimal places is fine) and also the output from U 100 110

7. (1 pt) Assemble and trace the following instructions:

```
mov ax, a0a0
rol ax, 1
mov ax, fffc
sar ax, 1
```

Trace the program with the T command. Try to explain the results after the rol and sar instructions. Submit a screen dump of the program trace and your explanations.

8. (3 pts) Compute the square root of an arbitrary16-bit integer at address [210]. The FILD instruction (Floating Integer Load) loads an integer from the specified address and converts it to IEEE extended double in ST(0).

```
FILD word  [210]
FSQRT
FSTP dword [200]
INT 20
```

After assembling the program, use the E command to enter data in memory. For example 900 decimal is 384 hex, but we have to enter in big-endian order (byte reversed)

```
-e 210
0C02:0210  66.84   0B.03
-
```

See next page–>

Here is a complete program run:

```
-a 100
0C00:0100 FILD WORD [210]
0C00:0104 FSQRT
0C00:0106 FSTP DWORD [200]
0C00:010A INT 20
0C00:010C
-e 210
0C00:0210  66.84   74.03
-g
Program terminated normally
-d 200
0C00:0200  00 00 F0 41 72 3B 63 3A-5C 4D 69 63 72 6F 73 6F   ...Ar;c:\Microso
0C00:0210  84 03 20 53 51 4C 20 53-65 72 76 65 72 20 32 30   .. SQL Server 20
0C00:0220  30 30 20 44 72 69 76 65-72 20 66 6F 72 20 4A 44   00 Driver for JD
0C00:0230  42 43 20 00 43 4C 49 45-4E 54 4E 41 4D 45 3D 43   BC .CLIENTNAME=C
0C00:0240  6F 6E 73 6F 6C 65 00 43-4F 4D 4D 4F 4E 50 52 4F   onsole.COMMONPRO
0C00:0250  47 52 41 4D 46 49 4C 45-53 3D 43 3A 5C 50 52 4F   GRAMFILES=C:\PRO
0C00:0260  47 52 41 7E 31 5C 43 4F-4D 4D 4F 4E 7E 31 00 43   GRA~1\COMMON~1.C
0C00:0270  4F 4D 50 55 54 45 52 4E-41 4D 45 3D 4E 41 53 52   OMPUTERNAME=NASR
-
```

Note that the result 00 00 F0 41 is really 41F0000 which is 30.0 (the sqaure root of 900).
For this problem:

a) (2 pts) Submit runs to compute the results decimal values 16 and 15129 and a disassembly (u 100 110) of your program

b) (1 pt) Submit a screen dump of the results and explain the rather strange results you get if instead of entering the values 84 then 03 in the e210 command you enter 03 and then 84.