Homework 1

Due: Tue, 08 Sep 2020 23:59:59 (approximately 93 days ago)
[Score: 17 / 20 points possible]
Weight = 1.0

The first part of this homework consists of some simple thought questions for which you can (mostly) look up the answers in the reference manuals or the lecture notes. It's best if you can use the reference manuals. The second part of the homework involves modifying an existing assembly language program to do what we're asking. You can use either the simulator or the actual microcontroller with System Workbench to do this exercise. (In either case, if you use comments, use only "//".)

Academic Honesty Statement [0 ... -20 points]

By typing my name, below, I hereby certify that the work on this homework is my own and that I have not copied the work of any other student (past or present) while completing it. I understand that if I fail to honor this agreement, I will receive a score of zero for the assignment, a one letter drop in my final course grade, and be subject to possible disciplinary action.

Raghuram Selvaraj ✓

Question 1 [1 point]

For the form of assembly language we use for this class, if you want the assembler to create an instruction that adds the contents of two registers and stores the result in a third register like this:

```
r0 = r3 + r5
```

as well as set the flags, how would you write that instruction so that the assembler would produce the right thing?

adcs r0, r3, r5 ✗

Question 2 [1 point]

For the form of assembly language we use for this class, if you want the assember to create an instruction that adds the contents of two registers and stores the result in a third register like this:

```
r1 = r1 + r6
```

but **not** change any flags, how would you write that instruction so that the assembler would produce the right thing?

adds r1, r6 ✓

Question 3 [1 point]

For either form of the add (immedate) instruction described in section A6.7.2 of the Architecture Reference Manual, what registers can be used for the source and destination? Express your answer as a comma-separated list of registers.

R0,R1,R2,R3,R4,R5,R6,R7 ✓

Question 4 [1 point]

Suppose you want to write an instruction that takes the value stored in register R1, add an immediate value to it and store it in R1. What is the largest immediate value you can use? Express your answer as a positive decimal number. (Hint: See A6.7.2 instruction encoding T2.)

255 ✓

Question 5 [1 point]

List the flags that may be affected by the instruction

`ADDS r1,r3,r5`

Express your answer as an unpunctuated list of flags in the order they appear in the xPSR. Read the documentation carefully, and try examples in the simulator. If you cannot think of a way to change the value of a flag, maybe it cannot be changed. (Write "none" if the instruction never changes any flags.)

Z ✗

Question 6 [1 point]

List the flags that may be affected by the instruction

`MOVS r1,r0`

Express your answer as an unpunctuated list of flags in the order they appear in the xPSR. (Write "none" if the instruction never changes any flags.) Read the documentation carefully, and try examples in the simulator.

Z ✗

Question 7 [1 point]

List the flags that may be affected by the instruction

`MOV r1,r0`

Express your answer as an unpunctuated list of flags in the order they appear in the xPSR. (Write "none" if the instruction never changes any flags.)

none ✓

Question 8 [1 point]

Is the following a valid ARM Cortex-M0 instruction?

`MOVS r0,#5`

(Answer yes or no.)

yes ✓

Question 9 [1 point]

Is the following a valid ARM Cortex-M0 instruction?

`MOV r0,#5`

(Answer yes or no.)

no ✓

Question 10 [1 point]

For the conditional branch instruction, **`B`**, what is the 4-bit code to use for "signed less than or equal"? Look at lecture 03 to understand how the conditional branch instruction works. (Express your answer in four sequential digits that are one or zero.)

1101 ✓

Question 11 [1 point]

What is the *stated* range, in bytes, for the permitted offset of a conditional branch? For instance:

```
BGE label1
```

Express your answer as a range of negative to positive values like: "-abc to xyz" (Hint: See A6.7.10 instruction encoding T1.)

-256 to 254 ✓

Question 12 [1 point]

What is the *stated* range, in bytes, for the permitted offset of an **unconditional** branch? For instance:

```
B label12
```

Express your answer as a range of negative to positive values. (Hint: See A6.7.10 instruction encoding T2.)

-2048 to 2046 ✓

Question 13 [1 point]

What instruction can *compare* values in two registers, R0 - R7, and set the flags to indicate what would happen if the value of one register was subtracted from the other? (i.e., this instruction does set the *flags* for subtraction without actually producing the 32-bit result of the subtract operation.)

CMP ✓

Question 14 [1 point]

What single instruction (that involves an assembler trick described in lecture) can you use to set register R4 to the value 0x87654321?

ldr r4, =0x87654321 ✓

Question 15 [1 point]

Write a single (logical) instruction that clears all bits in R0 that are set in R1 regardless of whether they were previously set in R0 or not. For instance, if the initial register values are

```
R0 = 0x00000000
R1 = 0x00008421
```

the resulting value in R0 after execution of the instruction should be 0. If, however, the initial register values are

```
R0 = 0xffffffff
R1 = 0x00008421
```

then the resulting value in R0 after execution of the instruction should be 0xffff7bde.

bics r0, r1 ✓

Question 16 [1 point]

Write a single (logical) instruction that sets the bits in R0 that are also set in R1 regardless of whether those bits were previously set in R0 or not. For instance, if the initial register values are

```
R0 = 0x44444444
R1 = 0x00008210
```

the resulting value of R0 after execution of the instruction should be 0x4444C654. If, however, the initial register values were

```
R0 = 0x44444444
R1 = 0x00000040
```

then the resulting value of R0 after execution of the instruction should be 0x44444444.

Try writing code in the simulator to set up the initial register values and the instruction you choose. Make sure it works.

orrs r0, r1 ✓

Question 17 [1 point]

Write a single (logical) instruction such that, given the following initial register values:

```
R0 = 0x12345678
R1 = 0xFFFFFFFF
```

the instruction would write the result 0xedcba987 to R0. And for the following initial register values:

```
R0 = 0x11111111
R1 = 0xFFFFFFFF
```

the instruction would write the result 0xeeeeeeee to R0. (Hint: Look at the values 0x12345678 and 0xedcba987 as binary values. How are they related?)

eors r0, r1 ✓

Program Modification [3 points]

Consider the code below that has many typographical errors:

```
.cpu cortext-m0
.thumb2
.syntax unify
.fpu softvfp

.globel main

main:
    movs   r0, #6
    move   r1, #20
    movv   r2, #0 // This is a loop counter
    mov    r3, #0 // This stores a sum.
loop:
    nop    // nop 1
    nop    // nop 2
    nop    // nop 3
    nop    // nop 4
    nop    // nop 5
done:
    bkpt   // Stop the debugger (breakpoint)
    b done
```

Modify this code, one step at a time. You can type it into the simulator and try it.

- First, fix the syntax errors! And don't laugh. These are the kind of errors you will make someday, so get used to finding and correcting them.
- Replace the first "nop" with an instruction that compares the values of registers R2 and R0. (i.e., it should set the flags as if it subtracted R0 from R2: (R2-R0))
- Replace the second "nop" with an instruction that redirects execution to the instruction following the "done:" label if and only if the result of the comparison in the previous instruction is greater than zero.
- Replace the third "nop" with an instruction that adds the value of register R1 to R3 and stores the result in R3.
- Replace the fourth "nop" with an instruction that increments the value of r2 by 1.
- Replace the fifth "nop" with an instruction that will send execution back to the instruction at the "loop:" label.

When the program reaches the

```
bkpt
```

instruction, the value of the R3 register should be 0x0000008c (decimal 140).

Important note: You should actually type in, run, and test this program. Then you should copy the whole thing (including the .cpu, .thumb, .syntax, and .fpu directives) from System Workbench or the simulator, and paste into the box below. There is a link below the text box that will invoke whatever you submit in the simulator. (Be sure to save it first.) That's what we will use to grade your work. If we can't assemble it, and see it run as specified, you don't get partial credit. It either works or it doesn't.

If you make updates in the simulator, remember to copy it back into the box below.

✓

```
.cpu cortex-m0
.thumb2
.syntax unified
.fpu softvfp

.global main

main:
    movs   r0, #6
    movs   r1, #20
    movs   r2, #0 // This is a loop counter
    movs   r3, #0 // This stores a sum.
loop:
    cmp r2, r0        // nop 1
    bgt done          // nop 2
    adds r3, r1       // nop 3
    adds r2, #0x1     // nop 4
    b loop            // nop 5
done:
    bkpt   // Stop the debugger (breakpoint)
    b done
```

Click here to try it in the simulator