

HOMEWORK ASSIGNMENT #2
Due Wednesday, September 30th, 2009

For each programming exercise, use the sample Keil project on the course web page as your starting point. Rename the *main.s* file to match the filename specified in the problem. Add your code to the file between the *main_loop* label and the **B main_loop** instruction. Do not alter the other source files. [Be sure to read through the documentation standards before beginning this assignment.]

1. (15 points) Instruction Encoding and Decoding

a. Determine what the corresponding assembly language instructions are for the binary machine language instructions below, and describe what the instruction will accomplish. Also write the corresponding RTL statement(s) indicating what the instruction does.

```
0xE0031002
0xEBFFFFFFA
```

b. For the following code fragment, explain how the instructions/pseudo-instructions are encoded to ARM7 instructions (i.e. identify what the value of each field of the instruction would be similar to chapter A4 of AARM, with a brief rationale for why that value is correct), and what binary value they would be encoded to.

```
ADD R0, R0, #-2
MyLabel
LDR R1, =0xFFFF70EF
LDR R2, =0xFFFFE07F
ADR R3, MyLabel
```

c. Students often write code like the following fragment. Why is this inefficient? Is it ever necessary to MOV an immediate constant into a register to be able to use that value in a data processing instruction (see page A3-9 of AARM for the list of data processing instructions), and if so, when?

```
MOV R0, #0x81
AND R1, R1, R0
```

2. (5 points) ARM7TDMI Flags

List the flag bits that are present in the CPSR, what each represents, and explain when they are updated. How do ARM7 instructions use the flags, other than directly accessing CPSR?

3. (10 points) ARM7TDMI Operating Modes

Explain what the different operating modes of the ARM7TDMI are, what causes the processor to enter each mode, how it exits that mode, and which registers (if any) are unique to each mode.

4. (20 points) Addressing Modes

Using the uVision3 sample project provided on the web page as a starting point, create a complete ARM program that has the following constants and variables. Do **not** make any changes to *aduc7026.s* or *exceptions.s*. Rename *main.s* to *teamX_hw2p4.s*, where X is your homework team number.

1. Declare a constant *ARRAY_LENGTH* to be equal to 3.

2. Allocate a half-word variable **hw16** in the code area, and initialize it to the binary value 1101001110011011. (Do not convert the value from binary.)
3. Allocate a byte variable named **b0** in the code area, and initialize it to 0x81.
4. Allocate an **ARRAY_LENGTH** byte array **b_array** in the code area, and initialize it to 1, -2, 3.
5. Allocate space in the data area for an array of **ARRAY_LENGTH** word variables named **w_array**.

In your program, write code to do the following (in sequence). Determine a reasonably minimal set of instructions to accomplish the required items, being sure to use the required addressing modes. (Deductions will be taken for excessive instructions.) **Do not use any branches in your code.** You may add constant data to store addresses. **Do not use any pseudo-instructions.**

6. Using indirect addressing with R9, do **byte** transfers (as many as required) to set each element of **w_array** to the value 0x01020304. Only set the value in R9 once, and then do not alter the value in R9 to complete this item.
7. Load the value in **hw16** to R10, but use two byte loads and any other required code to get the full 32-bit value. Assume that **hw16** contains an unknown signed value.
8. Load the value in **b0** into R11, assuming it is an unsigned value.
9. Load the value in **b0** into R12, assuming it is a signed value.
10. Write code that finds the minimum value stored in **b_array** and puts the sign-extended result in R7. (Assume that the values are stored in unsigned binary form, and that you do not know the values in the array!)
11. Write code that finds the minimum value stored in **b_array** and puts the sign-extended result in R8. (Assume that the values are stored in 2's-complement form, and that you do not know the values in the array!)

Comment *teamX_hw2p4.s* to clearly indicate the sections of code that correspond to each item in the list.

Important: Submit **ONLY** your program source code file *teamX_hw2p4.s* using the Homework #2 Dropbox in Learn@UW. Also, submit a **paper copy** of *teamX_hw2p4.s* with the rest of the assignment.

4. (15 points) JTAG Scan Interface

A. Describe the capabilities that the JTAG scan chain adds to the Texas Instruments 74BCT8245A as compared to Texas Instruments 74BCT245. List the 4 signals that form the test access port (TAP) and describe their functions. What is the TAP controller, and how is it operated?

Refer to IEEE standard 1149.1 and the datasheets for the devices. Sections 5-7 and figure B.10 in the standard are particularly helpful to look at, as well as the datasheets. The datasheets are available at <http://www.ti.com>. You can get that standard at IEEE Xplore going through the Wendt library site, or directly at <http://ieeexplore.ieee.org/Xplore/dynhome.jsp>. Going through Wendt ensures that you will not have access problems to the IEEE site.)

6. (15 points) Load/Store and Conditional Instructions

Using the uVision3 sample project provided on the web page as a starting point, rename *main.s* to *teamX_hw2p6.s*, where *X* is your homework team number. Declare two word arrays named *src* and *dest*, each of which contains space for 4 elements. The *src* array is to be declared in the code area, and should be initialized to the values 0x00010203, 0xFFFFEFD, 0x08090A0B, 0xFBFAF9F8. The *dest* array is to be declared in the data area, and should be uninitialized.

Write code that will copy the data from *src* to *dest*, to meet the following conditions. Assume that you do not have any prior knowledge of the source data values. You may use the LDR pseudo-instruction to load the array addresses.

1. Using *signed byte* load/store instructions, copy all data where the byte value is less than 2. If the byte value is greater than or equal to 2, place its 1's-complement in the destination byte instead.
2. Using the *load/store multiple* instructions, copy all data.

Separate the blocks of code, and clearly comment to indicate the code for a given task. **Use conditional instructions where necessary to make decisions - do not use any branches/loops in your code!** You can, of course, create a loop through all of your code to make repeatedly testing it easier. Your code should be reasonably efficient, within the constraints of the problem.

Important: Submit ONLY your program source code file *teamX_hw2p6.s* using the Homework #2 Dropbox in Learn@UW. Also, submit a **paper copy** of *teamX_hw2p6.s* with the rest of the assignment.

7. (10 points) More Conditional Instructions

Using only the instructions MOV, AND, OR, ADD, and ADC, write code fragments that perform the following tasks. **Use conditional instructions where necessary to make decisions - do not use any branches/loops in your code!** You can, of course, create a loop through all of your code to make repeatedly testing it easier. Your code should be reasonably efficient, within the constraints of the problem. Code submitted with syntax errors will receive no credit.

1. Count the number of 1s in the 4 least-significant bits of R0, and store the count in R1.
2. If the (unsigned) value in R0 is less than 256_{10} , multiply the value in R0 by 1.125_{10} .

8. (10 points) Exam Question

Design one original quiz question operating at Bloom's Taxonomy level 3 for any material covered in Module 2. This must test one of the module objectives in a specific problem. Explicitly show which particular objective you are attempting to test by quoting it. Provide a complete, detailed solution to your question.