

HOMWORK ASSIGNMENT #2
Due Wednesday, February 18th, 2009

*For each programming exercise, use the sample Keil project on the course web page as your starting point. (See the Using Keil uVision3 Projects link on the course web page for more info on using Keil projects.) Rename the **main.s** file to match the filename specified in the problem. Only add your code to the file between the **main_loop** label and the **B mainloop** instruction. Do not alter the other source files.*

1. (10 points) ARM7TDMI Operating Modes

What are the operating modes of the ARM7TDMI (list all 7), and for each operating mode give a short paragraph explaining the motivation for having such a mode (Why did ARM include these exception modes?). Also for each mode list any registers that are banked (unique copy for that mode only).

- 1) *User Mode* is the normal program execution mode for all application code.
- 2) *FIQ mode* is the fast interrupt mode. It allows fast context switches because it has local copies of R8-R14, and because its interrupt vector is located at the end of all interrupt vectors so the ISR can start without a branch
- 3) *Interrupt mode (IRQ)* is used for general purpose interrupt handling. LR and SP are banked registers
- 4) *Supervisor mode* is a protected mode for the operating system, and is the mode the processor starts in from reset. This mode has banked copies of the LR and SP.
- 5) *System mode* is a full privledged user mode for the operating system to execute code. It is not an exception mode, and therefore has no banked copies of the LR or SP.
- 6) *Abort mode* covers both exceptions of a pre-fetch abort or a data abort. This mode can be used for implementing paging virtual memory, or as a process memory protection fault mode. This mode has banked copies of the LR and SP.
- 7) *Undefined mode* is a mode entered when the processor encounters an undefined instruction. The main purpose of this mode is to allow the emulation of instructions. Particularly co-processor instructions if the co-processor is not present in the system. This mode has banked copies of the LR and SP.

ARM-state general registers and program counter

	System and User	FIQ	Supervisor	Abort	IRQ	Undefined
General registers	r0	r0	r0	r0	r0	r0
	r1	r1	r1	r1	r1	r1
	r2	r2	r2	r2	r2	r2
	r3	r3	r3	r3	r3	r3
	r4	r4	r4	r4	r4	r4
	r5	r5	r5	r5	r5	r5
	r6	r6	r6	r6	r6	r6
	r7	r7	r7	r7	r7	r7
	r8	r8_fiq	r8	r8	r8	r8
	r9	r9_fiq	r9	r9	r9	r9
	r10	r10_fiq	r10	r10	r10	r10
	r11	r11_fiq	r11	r11	r11	r11
	r12	r12_fiq	r12	r12	r12	r12
	r13	r13_fiq	r13_svc	r13_abt	r13_irq	r13_und
Program counter	r14	r14_fiq	r14_svc	r14_abt	r14_irq	r14_und
	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)
Program status registers	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

ARM-state program status registers

= banked register

2. (15 points) Instruction Encoding and Decoding

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

```
0xEBFFFFFFE ← BL -8
0xB0820311 ← ADDLT R0, R2, R1, LSL R3
0x00000000 ← ANDEQ R0, R0, R0
```

b. For the following instructions, explain how they can be encoded to ARM7 instructions, how the encoding would be done, what binary value they would be encoded to, and what limitations there are in each case.

```
MVN R7, #-268435441
```

Binary = E3E072FF MVN R7,#0xF000000F → Simply a FF with a ROR of 4-bits

```
MOV R9, #-1
```

Binary = E3E09000 MVN R9,#0x00000000 → Encoded with MVN because -1 (0xFFFFFFFF) cannot be encoded as 8-bit immediate with rotate, but the 1's complement of 0xFFFFFFFF can be encoded, and is simply 0x00000000

```
LDR R1, =0xFFC03FFF
```

Binary = E3E019FF MVN R1,#0x003FC000 → Loading a register with an immediate whos inverse can be encoded with the 8-bit immediate with rotation scheme. Assembler is smart enough to see this and replaces with a MVN statement.

```
LDR R2, =0xFF30CFFF
```

Binary = E59F20?? LDR R2,[PC,#0x00??] → This immediate cannot be achieved with a 8-bit immediate with rotation. So the assembler allocates and initializes a 32-bit word in Flash memory with 0xFF30CFFF and then uses PC relative addressing to load it.

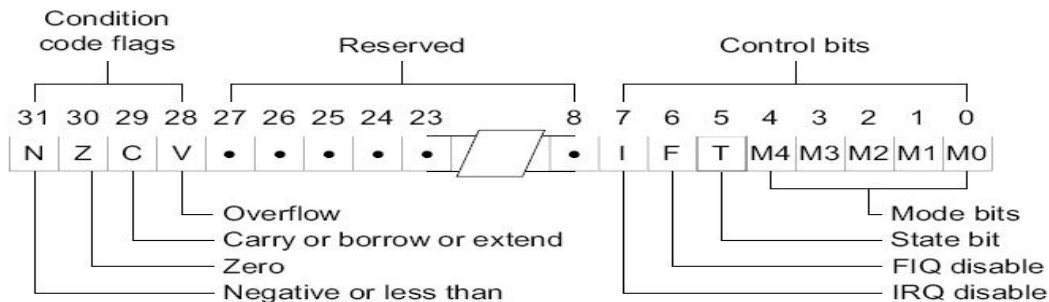
```
LDR R3, =(MyLabel)
```

Binary = E59F30?? LDR R3,[PC,#0x00??] → Similar to the one above. This time the assembler allocates a 32-bit word in Flash and initializes it with the address of my Label. Then it uses PC relative addressing to load that allocated word.

3. (5 points) ARM7TDMI Flags

List the flag bits that are present in the CPSR, what each represents, and explain when they are updated.

All ARM instructions are conditionally executed and can optionally update the condition flags. There are four condition flags present in the CPSR. The condition flags along with the other bit configurations of the CPSR are shown below.



Condition code flags

- N: Negative result from ALU
- Z: Zero result from ALU
- C: ALU Operation Carried out
- V: ALU operation overflowed

To cause the condition flags to be updated, the S bit of the instruction needs to be set by postfixing the instruction (and any condition codes) with an “S”. (Some instructions always update flags, i.e. CMP, TEQ, etc.).

For example to add two numbers and set the condition flags:
 ADDS r0, r1, r2 ; r0 = r1 + r2 and set flags

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 the *main.s* file to *teamX_main.s*, where X is your

homework team number. [Be sure to read through the documentation standards before beginning this assignment.]

- a) Declare a constant *ARRAY_LENGTH* to be equal to 8.
- b) Allocate a half-word variable *word16* in the code area, and initialize it to the binary value 1101 0011 1000 1011.
- c) Allocate a byte variable named *byte0* in the code area, and initialize it to 0xC3.
- d) Allocate a 10-element byte array *byte_array* in the code area, and initialize it to 1, -2, 3, -4, 5, -6, 7, -8, 9, -10.
- e) Allocate space in the data area for an array of *ARRAY_LENGTH* halfword variables named *halfword_array*.

In your program, write code to do the following (in sequence). Determine a reasonable (hopefully minimal) set of instructions to accomplish the required items, being sure to use the required addressing modes. You may add constant data to store variable addresses as required. **Do not use any pseudo-instructions. Use explicit PC relative addressing, and PC relative + DCD when necessary**

- f) Using indirect addressing with R6, do **byte** transfers (as many as required) to set the elements of *halfword_array* to the values 1, 2,4,8,16,32,64,128. Only set the value in R6 once, and then do not alter the value in R6 to complete this item.
- g) Load the value in *word16* to R7, but use two **byte loads** and any other required code to get the full 32-bit value. Assume that word16 represents a **signed** value.
- h) Load the value in *byte0* into R8, assuming it is an unsigned value.
- i) Load the value in *byte0* into R9, assuming it is a signed value.
- j) Write code that finds the minimum value stored in *byte_array* and puts the sign-extended result in R10. (Assume that the values are stored in **unsigned** binary form, and that you do not know the values in the array!)
- k) Write code that finds the minimum value stored in *byte_array* and puts the sign-extended result in R10. (Assume that the values are stored in **2's-complement** form, and that you do not know the values in the array!)

See link to file

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

Answer the following questions:

- a) How do the results of steps j & k differ? Why?

Result of the minimum is different because of how signed and unsigned numbers are represented. A -2 interpreted as an unsigned number looks like a large number.

Important: Submit ONLY your program source code file *teamX_main.s* using your homework team's dropbox in Learn@UW. Also, submit a **paper copy** of *teamX_main.s* with the rest of the assignment.

5. (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.)

The JTAG scan chain allows the 74BCT8245A to be used for both in-circuit automated board testing as well as debugging. The JTAG Test Access Port can “take snapshots samples of the data appearing at the device terminals or ... perform a self test on the boundary-test cells” (74BCT8245A datasheet). When connected in a chain with other JTAG-enabled devices, the JTAG master can run vigorous tests on the circuit board’s connectivity and on the functionality of the circuitry contained on the board.

The signals that form the test access port are TDI, TDO, TMS, and TCK. TCK is the test clock input, which is the master clock for all test logic. TMS is the test mode select input, which controls the operations run by the TAP. TDI is the test data input, into which flow instructions and data to the test logic. TDO is the test data output, out of which flow serial instructions and data from the test logic.

“The TAP controller is a synchronous finite state machine that responds to changes at the TMS and TCK signals of the TAP and controls the sequence of operations of the circuitry defined by [the IEEE 1149.1] standard” (IEEE 1149.1, 6.1.1). It is operated by feeding signals into TMS and TDI with TCK as the master serial clock.

B. Compare the SN74BCT8245A and SN54BCT8245A device characteristics. What are the key differences and similarities? What conclusions can come to about the two parts?

The SN74BCT8245A has higher output current than the SN54BCT8245A device, and the SN74BCT8245A also has shorter switching times. However, the SN74BCT8245A’s operating free-air temperature range is from 0°C to 70°C, whereas that of the SN54BCT8245A is -55°C to 125°C. Thus the SN54BCT8245A seems have been designed with extreme applications in mind (in fact, it is intended for military equipment), where the device must operate under wide temperature ranges. The SN74BCT8245A is meant for commercial applications, offering faster switching speeds in a more thermally regulated environment. The two devices are the same silicon, but the SN54BCT8245A specifications are derated due to the wider operating temperature.

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

Using the uVision3 sample project provided on the web page as a starting point, rename the *main.s* file to *teamX_hw2p6.s*, where *X* is your homework team number. Declare 3 word arrays named *src*, *dest1*, and *dest2*, each of which contain 4 elements. The *src* array is to be declared in the code area, and should be initialized to the values 0x00010203, 0x04050607, 0x08090A0B, 0x0C0D0E0F. 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 *dest1*, to meet the following conditions. When the byte value is less than 0x0A perform a straight copy of the byte. If the byte

value is greater than or equal to 0x0A, place its complement in the destination instead. 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.

- b) Write code that will copy the data from *src* to *dest2* using load/store multiple instructions. This time it is just a straight copy.

See link to source file

Separate the blocks of code, and clearly comment to indicate the code for a given task. **Make use of conditional execution in your code!**

Important: Submit ONLY your program source code file *teamX_hw2p6.s* using your homework team's dropbox in Learn@UW. Also, submit a **paper copy** of *teamX_hw2p6.s* with the rest of the assignment.

7. (10 points) Memory and Addressing Modes

- a) Refer to the code you wrote for problem 6 above. If you initialize the data in the *dest1/dest2* array, what happens? Does this make sense?

The *dest1/dest2* areas reside in SRAM. You cannot initialize SRAM in our simulator, or in real life. SRAM is random on powerup.

- b) B. Again referring to the code you wrote for problem 6 above, try to use the following pseudo-instructions to load the array addresses instead.
- ```
ADR R0, src
ADR R0, dest1
```

Explain what happened, and why it happened. If a pseudo-instruction works, describe in detail how it actually accomplished the load. If a pseudo-instruction will not work, explain in detail why it does not work (e.g. why is it not possible for it to work?).

The instruction "ADR R0, src" will complete the necessary operation. It will load the address of the *src* array into the register R0. This is accomplished by assembling an ADD instruction with PC as one of the operands. The starting address of *src* array is in the same area as the code and so the assembler can calculate a valid address using PC as one of its operand. The location is known RELATIVE to the PC.

However, ADR R0, dest1 cannot be assembled. A valid address cannot be computed in this case using the PC as one of the operands. This is because the address of the *dest* array is located in the data area, and the locations of the DATA and CODE areas are not known at assembly time.

### 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 state which particular objective you are attempting to test. Provide a complete, detailed solution to your question.