University of Pittsburgh
Department of Computer Science

CS2410 – Computer Architecture

Assignment #2
(Due on Wednesday October 5)

NOTE: (1) This is an individual assignment. (2) No late submission is accepted.

PART I [60]

Here is a *very* brief description of the "ACME" instruction set architecture. The focus of this part of the homework is on instruction encoding and pipeline design.

- ACME is a 32-bit load/store architecture.
- There are 16 32-bit general-purpose registers, R0~R15, and a status register SR.
- In general, instructions are 16 bits long ("instruction word" = 16 bits). Certain instructions may require additional 16-bit instruction word(s).
- Instructions are grouped into: Data transfer, arithmetic/logical/shift, control, and system. Typically, an arithmetic instruction involves one or two register operands. The following table captures the instruction set.

Group	**Instructions**
Data transfer	* Load and store instructions per data type ldb/stb Ry, @(Rx + D) ; load/store byte, D: an immediate displacement value ldh/sth Ry, @(Rx + D) ; load/store half ldw/stw Ry, @(Rx + D) ; load/store word * Data initialization for a register ld Rx, #Imm ; load immediate ld16 Rx, #Imm:16 (requires a second instruction word) ; load immediate ld32 Rx, #Imm:32 (requires a second/third instruction word) ; load 32-bit immediate mv Ry, SR ; Ry = SR (status register) mv SR, Rx ; SR = Rx

Arithmetic/logical/shift	* Arithmetic
	add Ry, Rx ; add, Ry = Ry + Rx
	addi Ry, Rx, #Imm ; add immediate, Ry = Rx + #Imm (signed)
	adc Ry, Rx ; add with carry, Ry = Ry + Rx + C
	sub Ry, Rx ; sub, Ry = Ry – Rx
	sbc Ry, Rx ; sub with carry, Ry = Ry – Rx – !C
	* Shift, n = 1, 4, 8, 16
	shl[n] Rx ; shift left by n bits, Rx = Rx << n
	shr[n] Rx ; shift right by n bits, Rx = Rx >> n
	shra[n] Rx ; shift right arithmetic by n bits, Rx = sext(Rx) >> n
	shlc[n] Rx ; shift left by n bits with carry, Rx = {Rx, C} << n

<table>
<tr><td></td><td>shrc[n] Rx ; shift right by n bits with carry, Rx = {C, Rx} >> n
rol[n] Rx ; rotate left by n bits
ror[n] Rx ; rotate right by n bits
* Logic
and Ry, Rx ; bit-wise and, Ry = Ry & Rx
or Ry, Rx ; bit-wise or, Ry = Ry | Rx
not Ry, Rx ; bit-wise not, Ry = !Rx
xor Ry, Rx ; bit-wise xor, Ry = Ry ^ Rx
andi Ry, Rx, #Imm
ori Ry, Rx, #Imm
xori Ry, Rx, #Imm
* Misc.
sextb Rx ; sign-extend byte
sexth Rx ; sign-extend half
* Compare
cmp Ry, Rx ; compare Ry with Rx and set the "test bit"
cmp Ry, #Imm ; compare Ry with #Imm and set the "test bit"
tst SR, #Imm:8 ; test bits (non-destructive bit-wise AND) in SR</td></tr>
<tr><td>Control</td><td>br <target> ; branch
brt <target> ; branch if the test bit is "true"
brf <target> ; branch if the test bit is "false"
bsr <target> ; branch subroutine
jmp Rx ; jump register
jsr Rx ; jump subroutine register</td></tr>
<tr><td>System</td><td>nop ; no-op
brk ; break
swi #Imm; software interrupt</td></tr>
</table>

(a) [30] Encode the instruction set. Show your encoding in a table. You will have to make assumptions here and there. For example, in certain cases, the width of the immediate field is not specified in the above table. In this case, assume a width for the immediate field and justify. Then, calculate the "unused" space in your encoding. Express the unused space in terms of # of new instructions you can add later, when the instructions are in a specific format (e.g., XXX Ry, Rx).

(b) [30] When the above ACME ISA was released within the company, an application engineer visited and asked you to add a few more "useful" instructions. They are (1) multiply (32 bits by 32 bits) and (2) a set of atomic memory bit read-modify-write operations: test-and-set, test-and-reset, and test-and-flip. These operations perform a byte read, test a specific bit in the byte, and update the specified bit in the memory in an atomic fashion. For example, test-and-set will retrieve a byte, move the specified bit value to the test bit in the status register, and set the specified bit in the memory by writing the properly modified byte back to the memory. The format of the test-and-set instruction is: tset @(Rx+D), #Imm:3. Note that with the 3-bit immediate value, you can select a bit in an 8-bit amount.

Your circuit designer told you that the multiply operation can take two cycles to complete (from multiplier input to output) but it can be pipelined. Besides, you realize that the result of 32x32 multiplication is 64 bits.

You also realize that atomic bit operations access memory twice (read and write).
Assuming a vanilla 5-stage pipeline (Fetch-Decode-eXecute-Memory-Writeback), discuss how these instructions can be accommodated, with detailed description of the potential complexities in pipeline management. Analyze the CPI impact of the new instructions. If needed, give example codes. Find unused space in the original map and encode these instructions. Show your encoding.

PART II [40]

Read the following two papers (posted on the course web page): (1) McFarling and (2) Smith and Pleszkun. Briefly summarize the papers first.

Discuss new (interesting) things that you learned.

Branch prediction and correct interrupt handling may add considerable complexity to (single) pipeline design. Explain how these will complicate superscalar processor design where there are multiple pipelines.

Submit your work at the class or directly to the mailbox of the instructor (box #276), located in the mail room on 5th floor, SENSQ.