CprE 381 Homework 5

[Note: This homework gives you some more practice with MIPS programming and thinking about testing your processor. As a happy coincidence, it will help provide a class-wide test program that I will share with you. Then the homework begins to cover processor design choices.]

1. Processor Implementation Details

   P&H(4.2) <§4.1>. The basic single-cycle MIPS implementation in Figure 4.1.2 (COD Figure 4.2) can only implement some instructions. New instructions can be added to an existingInstruction Set Architecture (ISA), but the decision whether or not to do that depends, among other things, on the cost and complexity the proposed additionintroduces into the processor datapath and control. The first three problems in this exercise refer to the new instruction:

   Instruction: **`JIC`** rt, imm
   Interpretation: PC = Reg[rt]+sign_extend(imm)

   a. Which existing blocks (if any) can be used for this instruction? Briefly describe them.
   b. Which new functional blocks (if any) do we need for this instruction? Briefly describe their functioning.
   c. What new signals do we need (if any) from the control unit to support this instruction? Include your control signal names (or widening of existing signals) and their specific assignment (including updated meaning of all values for widened control signals).

2. Processor Cycle Time Determination

   Assume the following latencies for the logic blocks in Figure 4.4.5 (COD Figure 4.17) from thetextbook.

I-Mem	Adder	MUX	ALU	Reg Read	D-Mem	Sign-Extend	Shift-Left-2	Control	ALU Control	AND gate
225ps	85ps	15ps	100ps	110ps	340ps	15ps	10ps	70ps	15ps	10ps

   a. Identify and quantify (i.e., give the path through the blocks and the time for that path) the worst-case path for each of the following: an arithmetic R-format instruction, a **`lw`** instruction, and a conditional branch instruction.

   [Note that in your project you will actually synthesize your processor and be asked to identifyits critical path—you'll find out the specific delays for the components you've designed when they are mapped to an FPGA. This problem should help you develop an intuitive sense of how to reason about critical paths.]

b. Rank the following design approaches in terms of which improve the cycle time the most. You **must** justify your ranking.
   i. Creating word addressable IMEM to eliminate branch / jump address shifting HW
   ii. Implementing quicker memory to have quicker DMEM access times
   iii. Designing a lower-latency control unit

3. Performance Analysis

You are in charge of selecting processors for computing simple Natural Language Processing (NLP) tasks. You are considering two processors, A and B (the only ones you and your roommates can afford) that have the following CPIs:

Instruction Type	**Cycles per Instruction**	
	Processor A	**Processor B**
Arithmetic, Logical, Shifts, Stores	3	3
Jumps	3	2
Conditional Branch	3	2
Loads	3	5

The following applications are the primary ones that you will need to run on your system:

```
# Application 1:
# This is an implementation of a single stop word index identification
# in a string.
# $a0 contains &string (string is an asciiz array of size string_size)
# $a1 contains &stop_word (stop_word is an asciiz array of size sw_size)
# $a2 contains string_size variable
# $a3 contains sw_size variable
  addiu   $t0, $0, 0
  j outer_cond
outer_loop:
  addiu   $t1, $0, 0
  jal inner_cond
outer_loop_cont:
  subu   $t2, $t1, $a3
  ori    $at, $0, 1
  sltu   $t2, $t2, $at
  addiu  $t0, $t0, 1
  beq    $t2, $0, outer_cond
  lui    $at, 0x1001
  ori    $a0, $at, 0x19
  addiu  $v0, $0, 4
  syscall
  lui    $at, 0
  ori    $at, $at, 1
  subu   $t0, $t0, $at
  addu   $a0, $0, $t0
  addiu  $v0, $0, 1
  syscall
  j exit
outer_cond:
  subu   $t2, $a2, $a3
  slt    $10, $8, $10
  beq    $t2, 1, outer_loop
  j exit
inner_loop:
  addu   $t2, $t0, $t1
  addu   $t2, $a0, $t2
  lb     $t3, 0($t2)
  addu   $t2, $a1, $t1
  lb     $t4, 0($t2)
  subu   $t2, $t3, $t4
  sltu   $t2, $0, $t2
  addiu  $t1, $t1, 1
  beq    $t2, 1, outer_loop_cont
inner_cond:
  slt    $t2, $t1, $a3
  beq    $t2, 1, inner_loop
  jr     $ra
exit:

# Application 2:
# This is an implementation of simple string tokenizer.
# $a1 contains string_size variable
# $a2 contains &string (string is an asciiz array of size string_size)
# $a3 contains &delimiter (delimiter is a byte array of size 1)
outer_cond:
  addiu  $t0, $t0, 1
  slt    $t4, $t0, $a1
  beq    $t4, 1, outer_loop
  j exit
outer_loop:
  addu   $t4, $t0, $a2
  lb     $t4, 0($t4)
  lb     $t6, 0($a3)
  subu   $t5, $t4, $t6
  ori    $at, $0, 1
  sltu   $t5, $t5, $at
  beq    $t5, 1, set_ending_index
  addi   $at, $0, 0
  subu   $t5, $t4, $at
  ori    $at, $0, 1
  sltu   $t5, $t5, $at
  beq    $t5, 1, set_ending_index
  j outer_cond
set_ending_index:
  addiu  $t3, $t0, 1
  addu   $t1, $0, $t2
  j inner_cond
inner_cond:
  slt    $t4, $t1, $t3
  beq    $t4, 1, inner_loop_print
  addiu  $a0, $0, '\n'
  addiu  $v0, $0, 0xB
  syscall
  j set_starting_index
set_starting_index:
  addu   $t2, $0, $t3
  j outer_cond
inner_loop_print:
  addu   $t4, $t1, $a2
  lb     $t4, 0($t4)
  addu   $a0, $0, $t4
  addiu  $v0, $0, 0xB
  syscall
  addiu  $t1, $t1, 1
  j inner_cond
exit:
```

a. Consider the two applications above. Calculate the average CPI for each application on each processor (two applications cross two processors means you should have 4 different CPI values). Assume **`string_size`** is always 20 in both applications and **`sw_size`** is always 3 (i.e., the stop word is "the") for application 1.

b. Which processor has better performance? *[Careful answering this part…it is a tricksy professor question.]* Provide the quantitative evidence of “better performance” and include a description of how you evaluated the two applications together. What would the relative frequencies have to be between the two processors in order for their performance to be identical (i.e., calculate the “breakeven frequency”)?