CS151B/EE116C - Homework #3, Due 4/24/2017

Reading Assignment:

- *Readings from #2 — Chapter 2: pp. 96-106; Appendix B: pp. B72-B74; Ch 5, **3rd Ed**: pp. 318 - 330, 331 - 339.*
- Chapter 1: pp. 28-40, 46-48 (performance)
- Chapter 4: pp. 272-288 (pipelining)
- Preparation for next week: Chapter 4: pp. 288-313

Problems:

(1) Consider the **single** cycle MIPS implementation with the datapath shown in Figure 4.17 (page 265) in the book and the control logic implemented based on the truth table in Figure 4.22 (page 269).

Your task is to modify this implementation to add support for the `jal` (jump and link) instruction, which is part of the MIPS ISA. This instruction is described on page A-63 in the book.

A) Show any required modifications to the datapath. You can show the modifications on a copy of Figure 4.17 (available on the class web page, *Useful Figures*).
Also explain the modifications in words, to make sure your modifications are clear.
In addition, if you need to modify any of the datapath building blocks, draw the implementation of the modified building blocks and explain the modifications in full detail.

B) Are any new control signals required? If so, list them with an explanation and identify them on the datapath diagram.

C) Show any necessary changes to the control logic by presenting a modified version of the control truth table in Figure 4.22 (available on the class web page, *Useful Figures*). For any entries that you add, mark all ''don't cares'' with an **X**.

(2) Write a recursive procedure in MIPS assembly language to compute the following function:

$$rfunc(n) = \begin{cases} 3 & \text{if } n \le 0 \\ 2 & \text{if } n = 1 \\ rfunc(\lfloor n/4 \rfloor + 1) + rfunc(\lfloor n/8 \rfloor - 1) & \text{if } n > 1 \end{cases}$$

Note that divides by powers of 2 can be implemented using shifts.
The argument n is in register \$a0 and the result $rfunc(n)$ should be returned in register \$v0.

Your implementation should follow the definition above in a straightforward way: for $n > 1$, your code should recursively call $rfunc(\lfloor n/4 \rfloor + 1)$ as well as $rfunc(\lfloor n/8 \rfloor - 1)$. Within this constraint, make your code as efficient as possible.

(3) Consider the **single** cycle MIPS implementation with the datapath shown in Figure 4.17 (page 265) in the book and the control logic implemented based on the truth table in Figure 4.22 (page 269). The implementation of the ALU in the datapath is shown in Figure B.5.12, page B-36.

Assume that in the original implementation of the ALU, the time (latency) to perform an AND operation is exactly the same as the time to perform an OR operation. Due to a change in the implementation, the time to perform an AND operation is doubled. We would like to predict how this change will affect the time it will take the processor to execute a program that includes many AND instructions and many OR instructions. The comparison is for the **same** program executing on the original implementation and the new implementation.

A) Is it possible that the change in the ALU will result in a change in program execution time? Your answer must be **yes** or **no**.

B) Explain in detail your answer to Part A.

C) Is it likely that the change in the ALU will result in a change in program execution time? Your answer must be **yes** or **no**.

D) Explain in detail your answer to Part C.

(4) Consider the multicycle MIPS implementation shown in the **3rd Edition** of the book, Figure 5.28 (page 323) and Figure 5.37 (page 338). Due to a hardware fault (malfunction), the MemtoReg output from the control unit is **always 0** (stuck-at-0).

Explain **in full detail** what will be the consequences of this change when the processor executes programs — how will it change the behavior of the processor **as observed by a user/programmer** who does not know and does not care how the processor is implemented internally? Be sure to clearly identify **each and every** consequence of this fault in as much detail as possible.

(5) Consider the multicycle MIPS implementation shown in the **3rd Edition** of the book, Figure 5.28 (page 323) and Figure 5.37 (page 338).

Your task is to modify this implementation so that it supports a new instruction, `swpc` (store word PC). The format of this instruction is exactly the same as the format of the `sw` instruction, except that the opcode is 010111.

The `swpc` instruction stores in memory the address from which the instruction is fetched. Thus, for example, if the `swpc` is at address 180, it will store in memory the 32-bit number 180 (note: **not 184**). The address into which the PC value is stored is computed in exactly the same way as the destination for the `sw` instruction (i.e., based on the Rs register and the immediate field).

You **cannot modify** the internal implementation of any of the datapath modules (building blocks). The only exception is that you can replace an existing MUX with a MUX that has more inputs (if needed, you must add the required additional control signals). You must note such a replacement of a MUX when describing changes to the datapath.

You cannot add any ALUs, adders, or subtractors to the datapath.

Your first priority is not to slow down any of the existing instructions. Your second priority is to minimize the changes to the datapath. Your third priority is to minimize the execution time of the new `swpc` instruction. Your fourth priority is to minimize the changes to the control. Note that you cannot speed up any of the building blocks used to construct the implementation in Figure 5.28. Furthermore, all the existing functionality must be maintained.

Note: if you need to add any new building blocks to the datapath or the control, it is **your responsibility** to make sure that it is completely clear **exactly** how each wire is connected to the new building block. If the new building block is not a copy of an already existing building block, you **must** draw the implementation of the new building block. (You do not have to show the implementation of standard MUXes). Make sure that your drawings are not messy.

A) Explain the basic idea of your modifications in 2-4 clear sentences.

B) Show the necessary modifications to the datapath by showing them on a copy of Figure 5.28 (from **3rd Ed**, available on the class web page).
Note: if you need to add any new datapath building blocks, show the implementation of the new building blocks in detail and explain their function.

C) Are any new control signals required? If so, list them with an explanation and identify them on the datapath diagram.

D) Modify the control unit state diagram (Figure 5.37 from **3rd Ed**) to reflect the added support for the `swpc` instruction.

E) With your implementation, how many cycles does it take to execute the new instruction?

Practice problems: You do not need to hand in a solution to the problems below.

(6) Write a procedure, `bfind`, in MIPS assembly language. The procedure should take a single argument that is a pointer to a null-terminated string in register `$a0`. The `bfind` procedure should locate the first `b` character in the string and return its address in register `$v0`. If there are no `b`'s in the string, then `bfind` should return a pointer to the null character at the end of the string. For example, if the argument to bfind points to the string "`imbibe`," then the return value will be a pointer to the third character of the string.
Note that each character of the string is stored in one byte and consecutive characters are stored in consecutive bytes in memory.

(7) Ackerman's function is defined by:

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0,\ n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0,\ n > 0 \end{cases}$$

Write a recursive procedure in MIPS assembly language to compute A(m,n). Your code **must** follow the standard MIPS calling conventions.
The argument m is in register $a0, the argument n is in register $a1, and the result $A(m, n)$ should be returned in register $v0.

(8) Consider the multicycle MIPS implementation shown in the **3rd Edition** of the book, Figure 5.28 (page 323) and Figure 5.37 (page 338). Due to a hardware fault (malfunction), there is a short circuit between the RegWrite signal and the most-significant bit of the ALUSrcB signal. As a result, whenever the control unit asserts (sets to 1) one of these signals, the other one is asserted as well. Hence, these signals are 0 only when the control unit sets **both** to 0.

Explain **in full detail** what will be the consequences of this change when the processor executes programs — how will it change the behavior of the processor **as observed by a user/programmer** who does not know and does not care how the processor is implemented internally? Be sure to clearly identify **each and every** consequence of this fault in as much detail as possible.

(9) Repeat Problem 5 above, but the new instruction to be added is `lui` (load upper immediate), described on page 112 in the class textbook.

(10) This problem is similar to Problem 5 above except that we wish to add support for four-operand arithmetic instructions such as `add3`, which adds three numbers together instead of two:

```
add3 $t5,$t6,$t7,$t8      # $t5 = $t6 + $t7 + $t8
```

Assume that the instruction set is modified by introducing a new instruction format similar to the R-format except that bits [0-4] are used to specify the additional register (we still use rs, rt, and rd) and of course a new opcode is used. Your solution should not rely on adding additional read ports to the register file, nor should a new ALU be used.

(11) On page A-57 there is a description of the `li` pseudoinstruction. Your task is to implement the functionality of `li` in a regular machine instruction on the multicycle MIPS. Since the immediate operand can be any 32-bit value, the instruction occupies two words. The first word contains the opcode and destination register number. The second word contains the immediate operand.

Pick a reasonable encoding for this instruction and repeat Problem 5 for `li` instead of `swpc`.

(12) Consider the multicycle implementation shown in Figure 5.28 (page 323, **3rd Ed**).

Your task is to modify this implementation to implement a procedure call mechanism. This mechanism will not be the one defined as part of the MIPS ISA. Instead, it will be based on using the stack. Register 29 will be used as the stack pointer. The stack grows down and uses the ''last full'' convention.

The semantics of the `jal` instruction will be changed. The new `jal` will push the return address on the stack instead of saving it in register 31. The encoding of the `jal` instruction remains the same.

Your first priority is to maximize the performance of the new `jal` instruction without slowing down any of the existing instructions. Your second priority is to minimize the changes to the datapath and control. You cannot speed up any of the building blocks used to construct the implementation in Figure 5.28. All the existing functionality must be maintained.

A) Show the necessary modifications to the datapath. If modifications are required, show the entire modified datapath (Figure 5.28 modified as necessary). If you need to modify any of the datapath building blocks, draw the modified building blocks and explain the modifications.

B) Are any new control signals required? If so, list them with an explanation and identify them on the datapath diagram.

C) Modify the control unit state diagram (Figure 5.37) to reflect the added support for the new `jal` instruction.

D) In order for this mechanism to be useful, is it necessary to implement a new `return` instruction? If so, explain why. If not, show how to implement a return from a procedure without a special new instruction.

(13) Consider the multicycle MIPS implementation described in chapter 5, **3rd Ed**. The propagation delays for the major components in the datapath are as follows:

Component	Delay
ALU	3ns
Memory	5ns
Register File	6ns

All other delays are negligible.
What is the maximum possible clock frequency (in Hz) of the system clock. Explain your answer.