Final

CSEE W3827 - Fundamentals of Computer Systems — Dec. 17, 2009
Fall 2008 — Prof. Rubenstein

This final contains 5 questions, and totals 100 points. To get full credit you must answer all questions. **BOOKS AND NOTES ARE PERMITTED, BUT ELECTRONIC DEVICES ARE NOT!** The time allowed is 2 hours, 50 minutes.

Please answer all questions **in the blue book**, using a **separate** page for each question. **Show all work!** We are not just looking for the right answer, but also how you reached the right answer.

1. (10 pts) Suppose that the stack is given its own memory and **direct-mapped cache**. Show that blocksize does matter as follows:

   For simplicity, assume the top of the stack starts at memory location 0. Each time an item is pushed onto the stack at memory location i, the top of the stack becomes memory location $i + 1$.[1]

   Assume that the cache holds only 4 memory words.

   (a) (3 pts) Give a sequence of pushes and pops where there are more cache hits when block size is 4 as opposed to when it is 1.

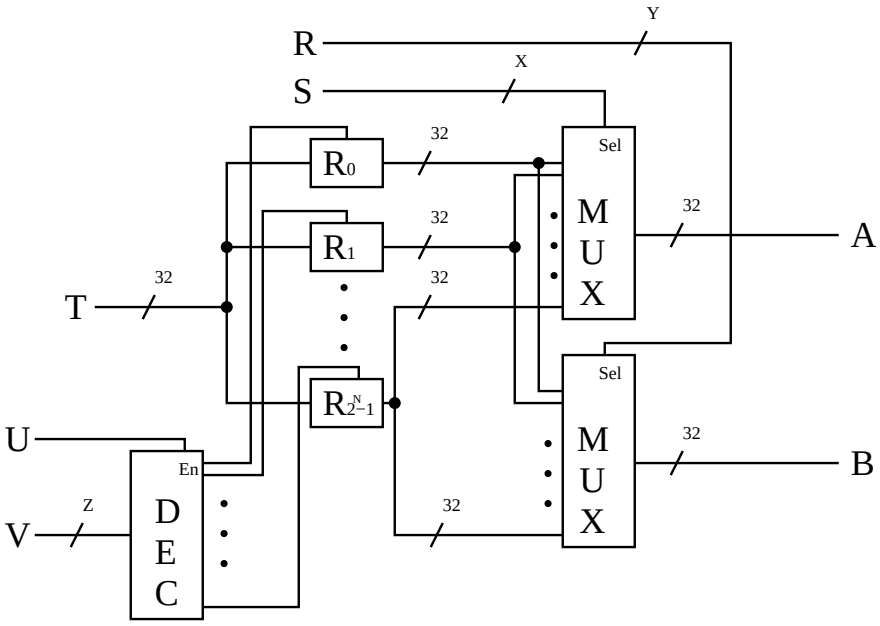
   (b) (7 pts) Give a sequence of pushes and pops where there are fewer cache hits when the block size is 4 as opposed to when it is 1.

   Write your sequence using 'H' for push and 'P' for pop, e.g., HPHHPP is the sequence push, pop, push, push, pop, pop. Put an arrow under each cache hit. For instance:

instruction #:	0	1	2	3
	H	P	H	H
		↑		↑

   implies that the first push was a miss, but that the remaining three pops and two pushes are hits. Note that the above example is only intended to demonstrate the notation - there is no claim that the shown hit pattern is demonstrative of any particular block size.

[1] This makes the problem slightly easier than when the top of the stack is at some memory address A and pushing an item shifts the stack's location from its current i to $i - 1$.



2. (20 pts) The above figure depicts a standard register file with 2^N registers labeled R_0 through R_{2^N-1}. Each register has a 32-bit input (on the left) from which it can read and store new data when its enabler (on top of the register) is set to 1, and a 32-bit output (on its right) that reveals its stored 32-bit value.

   The register file also contains 2 multiplexers, and a decoder with enable to assist configuring the register file to read from and write to the desired registers.

   Assume that at the end of clock cycle j, register R_i stores the value $i + 10$ for all $0 \leq i < 2^N$.

   (a) (1 pt) What are the values of X, Y, and Z (the number of bits on the wires that forward inputs S,R, V to the circuitry)?

   (b) (3 pts) Suppose during clock cycle $j + 1$, all input bits are 0, i.e., $R = S = T = U = V = 0$. What are the new values of all the registers at the end of the cycle, and what are the values of outputs A and B?

   (c) (3 pts) Same question for clock cycle $j + 1$, except that **all** bits of S, U, and V are 1s (R, T still all 0's).

   For the next two parts, we want to modify the register file so that two standard MIPS instructions, I_0 and I_1 are able to simultaneously access the single register file during the same clock cycle. Assume that in addition to the 5 inputs R, S, T, U, V and 2 outputs A, B that exist with the current register file for I_0, an additional 5 inputs inputs R_1, S_1, T_1, U_1, V_1 and 2 outputs A_1, B_1 are added.

   (d) (5 pts) To enable simultaneous **read access** of the register file (e.g., if I_0 is BEQ R_0, R_3 and I_1 is BEQ R_2, R_3, then values in registers 0 and 3 should be read for I_0 and values in registers 2 and 3 should be read for I_1), **explain in one or two sentences** what circuitry and wiring needs to be added and/or modified.

   (e) (8 pts) Suppose if both instructions require write access, then only I_0 is granted write access (e.g., if I_0 is ADD R_0, R_1, R_2, and I_1 is ADD R_4, R_5, R_6, then only I_0 writes to the register file, even though there were no register conflicts), **Draw** the circuit where you would make additions and modifications. DO NOT DRAW THE PARTS OF THE CIRCUIT THAT YOU WOULD NOT MODIFY. (Hint: some, but not much of the existing circuit needs to be modified).

3. (20 pts) In the traditional MIPS architecture, the loadword (lw) command lw R_B, $C[R_A]$ adds a constant C to the value in the "A" register to determine the memory address to load into the register file. Similarly, storeword (sw R_B, $C[R_A]$) adds a constant to the value in the "A" register to determine the memory address to which the value in the "B" register is written. In this problem, we will design a modified architecture, MIPS-NEW, where:

   - lw is of the form lw R_B, R_A, C obtains the memory address from which to read from the "B" register and stores this value plus constant C in the register file, e.g., for lw R_1, R_2, C, if the value in R_2 is x, then a value y is obtained from memory address x and $C + y$ is stored in R_1.
   - sw is of the form sw R_B, R_A, C, and adds a constant to the value in the "A" register and stores this summed value in the memory address described by the "B" register, e.g., for sw R_1, R_2, C, if the value in R_1 is z and the value in R_2 is y, then $z + C$ is stored in memory address y.

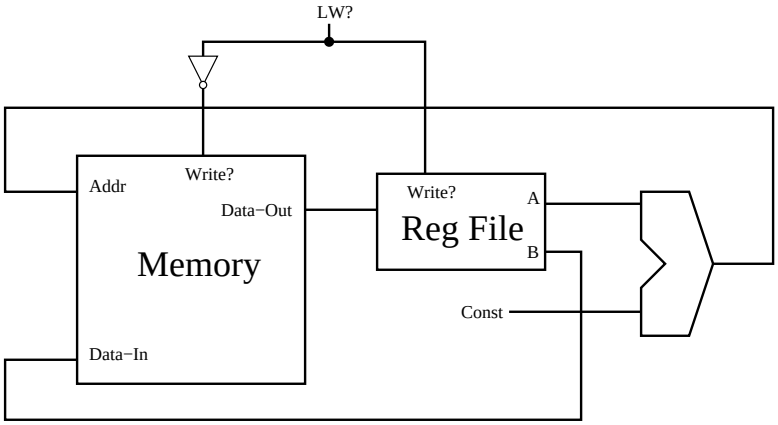


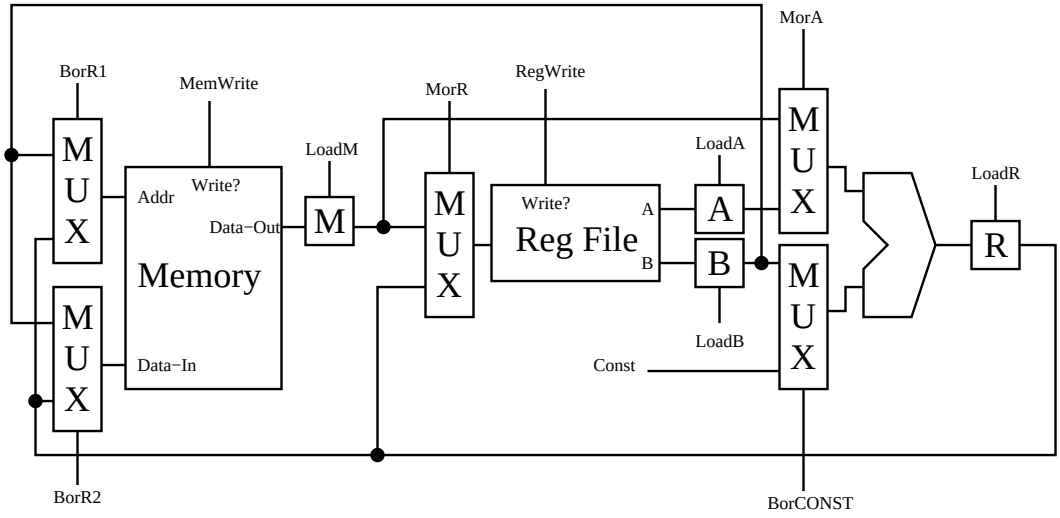
Figure 1: Simplified Single-cycle Architecture

Figure 1 shows a stripped-down diagram of a **traditional** MIPS single-cycle architecture. Only the essentials needed to implement lw and sw are shown. There is a single flag input LW, which equals 1 for a lw, and 0 for a sw. You may assume that the ALU always adds. In a similar style of diagram, draw the MIPS-NEW architecture. (Hint: you may use an additional ALU to implement both lw and sw).

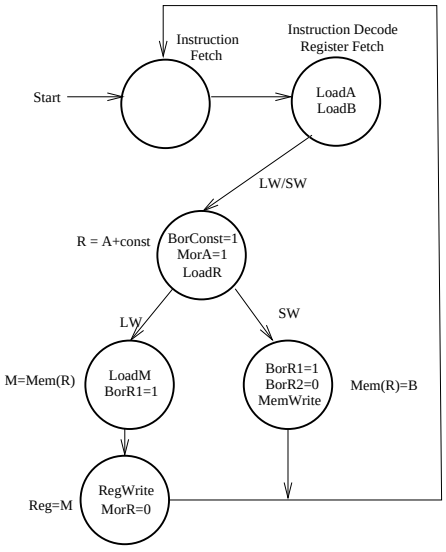
4. (25 pts) Figure 2(a) shows a stripped-down diagram of a MIPS multi-cycle architecture. M, A, B and R are all registers that are respectively loaded with the output from memory, the "A" register, the "B" register, and the ALU when their respective Load input is enabled (set to 1). Setting RegWrite and MemWrite to 1 respectively enables the register file and memory for writing. The remaining inputs are of the form YorZ and each feeds into a multiplexer. Setting YorZ to 0 selects Y, and setting it to 1 selects Z.

   Figure 2(b) depicts the state machine that uses this architecture to implement the **traditional** MIPS lw and sw. Note that each state is labeled on the side by what the architecture does when in that state, and how the flags are set is written inside the state.

   Your task is to draw the state machine that implements the MIPS-NEW architecture.



(a) Architecture



(b) State Diagram

Figure 2: Multi-cycle

5. (25 pts) In this problem, you are to design the pipeline for a pipelined architecture that implements MIPS-NEW (at a very high level).

   Each stage of pipeline is annotated: **I** for instruction fetch, **R** for register fetch, **X** for Execute, **M** for Memory, and **W** for writeback; the annotation describes the general type of operation performed in that stage. We can simply write the stages in the order that they appear. For instance, the **traditional MIPS** pipeline ordering is IRXMW.

   We use parentheses to indicate the combining of two stages into a single stage. e.g., I(RX)MW is a 4-stage pipeline with register fetch and execute (ALU) operations both performed in the second stage (in parallel, since both cannot be performed in sequence) within a single clock cycle). Note that for some parts below, the same operation may be required in two different stages, e.g., I(RX)(RM)W, indicates that we fetch from registers during stages 2 and 3 (while executing and accessing memory in parallel).

   Assume that the lw and sw instructions discussed below are the only instructions that access data memory.

   (a) (4 pts) Before building MIPS-NEW, assume we wish to modify the memory access instructions to the form lw R_B, R_A and sw R_B,R_A, so that there is no constant: sw transfers the value stored in R_B into the memory address indicated by register R_A, and lw transfers the value in the memory address specified by R_A into R_B. Which two stages of the pipeline can be combined, and why? (just 1 or 2 sentences please)

   (b) (2 pts) Does combining these two stages have a significant impact on the speed (e.g., instructions per second) of a really long program? Why or why not?

   (c) (3 pts) Suppose we now modify lw to behave as described by MIPS-NEW (sw behaves as in part 5a). What is the best order in the pipeline for this lw instruction?

   (d) (3 pts) Suppose we allow sw only to behave as described by MIPS-NEW (lw behaves as in part 5a). Now what is the best order of the pipeline?

   (e) (3 pts) Suppose both lw and sw are design as described by MIPS-NEW. Which operation (I,R,X,M,W) must be implemented in two separate stages of the pipeline (which letter appears twice)? Why?

   (f) (10 pts) Select a particular ordering of the stages that can implement both lw and sw instructions. For your staging, indicate the various hazards that would occur for the following code:

   lw R_1, R_2, 5
   add R_3, R_1, 10
   sw R_3, R_1, 0

   Suppose circuitry to do forwarding is added. Considering each hazard independently (pairs of instructions), indicate how many cycles the subsequent instruction that caused the hazard must be stalled