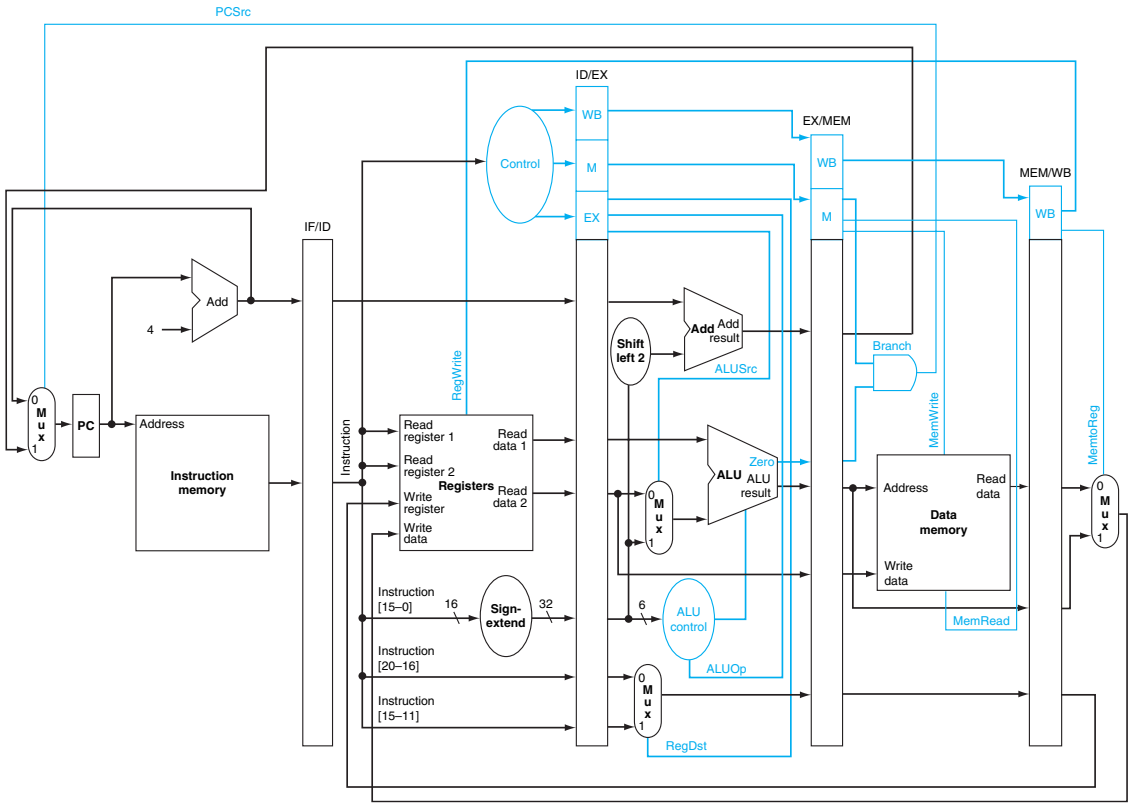
CSEE 3827: Homework 6 — Due: Wednesday 5/5/10

This homework is due by 1pm on Wednesday 5/5. To turnin your homeworks you may either scan and upload to courseworks, or hand-deliver it to Prof. Kim's office in 469 CSB.

1. To date we have not explicity stated the size (in number of bits) of the pipeline registers in our 5-stage MIPS pipeline. In the pipelined processor shown below, calculate the bitwidth of the four pipeline registers: IF/ID, ID/EX, EX/MEM, and MEM/WB. Be sure to include the necessary control signals in your final totals. NB: This version of the processor does not have support for operand forwarding or stalls.



2. Structural, data and control hazards typically require a processor pipeline to stall. Listed below are a series of optimization techniques implemented in a compiler or a processor pipeline that are designed to reduce or eliminate stalls due to these hazards. For each of the following optimizations, state which pipeline hazards it addresses and how it addresses it. Some optimization techniques may address more than one hazard, so be sure to include explanations for all addressed hazards.

   (a) Branch Prediction
   (b) Bypass/forwarding Logic
   (c) Instruction Scheduling
   (d) Increasing the number of functional units (ALUs, adders, etc.)
   (e) Caches

3. Consider the following code sequence:

```
add $t0, $a0, $s0
lw $t1, 0($t0)
add $t1, $t1, $t1
sw $t1, 0($t0)
```

(a) Assuming a MIPS processor that has the standard 5-stage pipeline with the ability to stall (i.e., insert bubbles in the execute stage) but has no forwarding paths whatsoever. Complete the execution schedule on this processor for the above code snippet as begun below. You may assume all memory accesses complete in a single cycle.

Instr	cc_1	cc_2	cc_3	cc_4	cc_5	cc_6	cc_7	cc_8	cc_9	cc_{10}	cc_{11}	cc_{12}	cc_{13}	cc_{14}	cc_{15}	cc_{16}
`add`	IF	ID	EX	MEM	WB											
`lw`																
`add`																
`sw`																

(b) Assuming now that the processor does have full forwarding hardware (i.e., can forward from MEM to EX and from WB to EX) what would the schedule look like?

Instr	cc_1	cc_2	cc_3	cc_4	cc_5	cc_6	cc_7	cc_8	cc_9	cc_{10}	cc_{11}	cc_{12}	cc_{13}	cc_{14}	cc_{15}	cc_{16}
`add`	IF	ID	EX	MEM	WB											
`lw`																
`add`																
`sw`																

(c) Would there be any performance benefit if the programmer substituted `sll $t1, $t1, 2` for the second `add` instruction above? Why or why not?

4. Assume a stack of 32-bit integers, where the base address of the stack is `0x00006000`.

(a) Give a list of the six addresses that are accessed by the following sequence of operations:

```
stack.push(1)
stack.push(2)
stack.push(3)
stack.pop()
stack.pop()
stack.pop()
```

(b) Illustrate the state of a 64-byte, direct-mapped cache after these 6 memory operations. The cache can hold 8 8-byte lines (i.e., 1 line = 2 words). You may assume the cache is empty at the start. Be sure to show the state of the valid and tag bits in addition to the data values.

(c) What is the miss rate? The hit rate?