EE457 MT (~20%)
Closed-book Closed-notes Exam; No cheat sheets; Calculators are allowed. Verilog Guides are allowed but not needed.

Spring 2015
Instructor: Gandhi Puvvada

Friday, 3/27/2015
09:00 AM - 12:00 Noon (3 Hour 00 min. = 180 min)
Location: THH301

Student’s Last Name: _______________________________________
Student’s First Name: _______________________________________
Student’s DEN Bb username: ______________________________@usc.edu

Ques# Topic Page# Time Points
1 2nd edition multi-cycle CPU 2-4 40 min. 48
2 Lab 7 Part 1 3-elem. adder pipeline 5-7 50 min. 69
3 5-stage Early branch and Branch Delay slot 8-9 50 min. 59
4 convert 5-stage early branch to 3-stage 10-11 20 min. 35
5 Cache (reverse engineering) 12 15 min. 36

Total Cover +11 175 min. 247

Perfect Score 230

Viterbi School of Engineering
University of Southern California
Improve the 2nd edition multi-cycle CPU:

Reproduced on the next two pages are the 2nd Edition CU (Control Unit) state diagram and the DPU (Data Path Unit). Miss Trojan suggested that, instead of returning to state 0 and then fetching the next instruction, you can prefetch the next instruction in the last clock of the current instruction and return to state 1, there by saving a clock. She wanted to do this in the case of a lw (load word) instruction, a R-Type instruction and a jump instruction. She was about to copy all the signals of the state 0 into the three states, 4, 7, and 9, but realized that she has to do something special in the case of a J (a JUMP instruction) (state 9). She added a mux next to the PC in the DPU and named the select line \textbf{PFCJ (PFCJ = PreFetch Control for JUMP)} and made it a "1" in the state 9 as shown.

She left the design for you to complete in the next two pages.
(a) add "\textit{PFCJ = 1}" or "\textit{PFCJ = 0}" in a subset of the remaining 9 states where it matters.
(b) add new state transition arrows as needed to return to state 1 and delete some earlier state transition arrows returning to state 0.
(c) In the datapath, complete the connections to the new mux and break or make any other connections as needed.
(d) adjust any other signals such as PCSource.

1.1 State briefly, what is the difference between prefetching the next instructions from states 4 and 5 as compared to prefetching the next instructions from state 9?

1.2 Why didn’t Miss Trojan try to prefetch the next instruction from states 5 (last clock of SW) and state 8 (last clock of beq)? Fetching or prefetching of an instruction requires two important resources: 1. Memory to fetch from and 2. ALU to increment the PC. In state 8 (last clock of beq), the ALU is busy checking equality for beq and hence not available for PC incrementation.

1.3 Why similar prefetching is not possible in the first edition design of the multi-cycle CPU?

Basiclly, in the first edition design, in the absence of MDR, memory continues to read its DATA in state 4 and is not available to read the next instruction. Similarly, in the absence of an ALU output register, the ALU continues to calculate the data for the R-Type, in state 7. It is however possible to implement this improvement for the JUMP instruction, as both the ALU and the memory are available.
Adjust control signals for state 9. Add or delete state transition arrows. Write "PFCJ = 1" or "PFCJ = 0" where needed.

**Note:** Exception handling becomes a little complex with the way we were prefetching here. Note: Perhaps you have noticed that we could not perform prefetch in state 5 because the single-ported memory is busy with SW and also we could not prefetch in state 8 because ALU is busy performing equality check for br eq.
Problem with the above design is that the PC [31:28] was tapped after the multiplexer controlled by the PFCJ to form the JA (the Jump Address. So when (PFCJ == 1), we created a combinational feedback on the PC[31:28] (without any driver for the 4-bit signal). The simple fix is shown on the next page.
Complete connections to the new mux. Make other needed adjustments.

Notice that the PC[31:28] is now tapped from the upstream of the mux governed by PFCJ. So the PC[31:28] is now part of the Jump instruction's PC plus 4 as needed by the Jump instruction.
Lab 7 Part 1 3-element adder pipeline

Our VLSI engineer, Miss Bruin, was doing VLSI layout for our 5-stage pipelined 3-element adder (Lab 7 Part 1) and ended up adding a dummy stage between the original EX1 and EX2. So now we have a 6-stage pipeline with EX1, EX2 (the dummy stage) and EX3 (the original EX2). A Z_Mux was added to the dummy stage on the next page. Complete the design on the next two pages.

Before you stall an instruction in ID stage, you make sure that the instruction you are stalling is not a NOP. T/F

Is this the same in Lab 6 5-stage pipeline discussed in the book/class? Y/N

Reason for similarity or difference: In Lab 7, the opcode is just single run bit, so no time is lost in decoding. Hence it was used in the STALL logic without elongating the clock. In Lab 6, HDU does not (but not - do does) do so for the fear of elongating the clock.

Stalling: Spurious stalls lower performance (lower performance) produce wrong results.

Forwarding: You care to check if the senior providing forwarding help is not a NOP. T/F

You care to check if the junior receiving forwarding help is not a NOP. T/F

Is one of the above very important? Explain: If we do not check and (ID/EX.WriteRegister /= 0) if the senior is not a NOP, then the program results are wrong!

Can you stall the dependent instruction in EX1 stage instead of in the ID stage either in the original 5-stage pipeline or in this 6-stage pipeline? No. In either case, we can stall in the EX stage as we have multiple sources out of which one can be dependent on the WB stage.

Dependency for the Z register on a senior didn’t (did / didn’t) lead to a stall in the original 5-stage pipeline. Dependency for the Z register on a senior didn’t (did / didn’t) lead to a stall in this 6-stage pipeline.

For the sake of this exercise, is it possible to move one or two or three Z muxes from their current position to another stage? For each Z mux, state your answer using words such as possible or not possible and also desirable or not desirable. Please notice the 3 forwarding paths. Since each of them are receiving help from their respective senior in WB stage, they cannot be moved towards the right side of the diagram.

A similar to the "lw" delay slot, can you think of a delay slot in this design. If so how many delay slots? One or two or three? How does it affect your hardware design and cost? If the compiler designer is not quite smart and fills the delay slot using a NOP 90% of the time, is that worse than the hardware solution or still better? How? "lw" delay slot tries to avoid dependency stall by separating (distributing) the dependent inst. Here the worst case of the compiler solution (just 100% more) can be slightly worse than the HW solution because the dependent inst. Here the worst case of the compiler solution (just 100% more) can be slightly worse than the HW solution because the dependent inst. Here the worst case of the compiler solution (just 100% more) can be slightly worse than the HW solution. Here the senior instruction may turn himself into a NOP because of overflow in EX1. If such senior instructions are many then HW can be better. Based on the STALL logic, up to 2 delay slots can be declared. Delay slots decrease the cost of HW as STALL logic is not needed.
Complete the design (6 EN controls, RegFile write-port connections, forwarding paths, bubble-injection, etc.)

Generate on the next 2 pages
STALL, X_FORW1, Y_FORW1, Z_FORW1, Z_FORW2, Z_FORW3.

Pipelined 3-element Adder
Block Diagram
LAB 7 Part1 with a dummy stage
All forward controls (X_FORW1, etc.) are produced when the receiving instruction reaches the stage, where the forwarding MUX is located.
Graders: Here I am showing an alternative design where all forward signals are produced in the ID-stage and carried into the stage where the forwarding mux is located. For students to get credit they should show the FFs or FFs, either here or on the block diagram.

Notice that we are using OUT signals, so as to make sure that the signal has not turned himself into a NOP!
Early branch and Branch Delay slot

The Block diagram for the Early Branch design from our lab 6 is given on the next page with support for JUMP instruction and one delay slot added as per the solution to question Q3.4.1 of Midterm Fall 2013.

3.1 Did we support delay slot for beq only or jump only or for both? **Both**

Explain: *Since we do not flush the IF stage instruction, the delay slot applies to all control instructions and *etc.* executing from the ID stage.*

3.2 IF.Flush is crossed off because we do not want to flush the IF stage instruction (because we are supporting an delay slot).

Why VDD is connected to the wrist-band FF here? Can we connect GND (ground) instead? Can we remove the wrist-band FF all together? Does connecting the GND instead of VDD call for change in the Verilog code for the Wrist-Band FF? **Yes** (Yes/No). Any changes to the block diagram needed if we change VDD to GND?

We can't remove the wrist-band FF altogether, as during reset (R = 0) you want to convert the RANDOM inst in ID stage to a bubble. This inverter at the CU tells us that a zero in the wrist-band FF implies flushing the instr in the ID. Hence the VDD makes sense after reset, no more flushed.

If we remove the VDD connection and restore the crossed-off part, what happens to beq instruction and what happens to the j (jump) instruction? Would you suggest doing something more? Explain. If you restore, then successful branch flushes IF, which means that there is no delay slot for beq.

If there is no delay slot, both jump and successful beq in the ID stage should cause flushing of the instruction in the IF stage. This is achieved through the OR gate shown on the next page. In the absence of this OR gate and with IF.Flush restored, it is as if the ISA has declared a delay slot for the JUMP only but not for the beq (which is very uncommon). If you need to add or delete or modify something, please show the same on next page.

3.3 Why are we clearing ID/EX, EX/MEM, MEM/WB stage registers on reset using ?

To make sure that on reset effectively a bubble prevails in the EX, MEM, and WB stages (instead of some random instruction).

3.4 In a 7-stage pipeline (in one of the questions in the lab 6 part 4), as shown on the right, we made arrangements for instruction(s) in each of IF1 and IF2 (each of IF1 and IF2 / only IF1 / only IF2) to wear a wrist-band, and (and / but not ) carry it until it/they reach(es) the ID stage.

Modify the diagram on the right to support 2 (two) delay slots. Explain: Two delay slots do not flushing of IF1 and IF2. However you need the FF with VDD input so that on reset you have bubbles in IF2 and IF1.

It is a SIN (OK / a SIN) to fill the delay slot of a branch with another branch instruction. Branch delay slots are filled by the compiler at compile time and not by hardware at run-time.

On RESET, the two FFs are cleared and after reset is over; the zero in the IF1/IF2 wrist-band FF moves to the FF in IF2/ID, which means the bubbles are trickling out and after that there is no flushing even if there is a successful branch.
3.2 IF. Flush is crossed off because we do not want to flush the IF stage instruction (because we are supporting one delay slot).

Why VDD is connected to the wrist-band FF here? Can we connect GND (ground) instead? Can we remove the wrist-band FF all together?

Does connecting the GND instead of VDD call for change in the Verilog code for the Wrist-Band FF?  **Yes**  (Yes/No). Any changes to the block diagram needed if we change VDD to GND?

We cannot remove the wrist band FF altogether, as during reset ($R = 0$) you want to convert the RANDOM inst in ID stage to a bubble.

This inverter at the CU tells us that a zero in the wrist band FF implies flushing the inst in the ID. Hence the VDD makes sure, after reset, no more flushing.
If there is no delay slot, both jump and successful beq in the ID stage should cause flushing of the instruction in the IF stage. This is achieved through the OR gate.

Say, the cross-off and vdd are removed. It means we do not have a delay slot. Then the newly added support for the jump instr. should include flushing.
4  (35 points) 20 min.  5-stage => 3-stage

We want to convert our 5-stage early branch pipeline to a 3-stage early branch pipeline by combining the IF and ID stages into one IF_ID stage and also combining the MEM and WB stages into one MEM_WB stage. I have reproduced on the next page our 5-stage pipeline and just removed the IF/ID stage register and the MEM/WB stage register but did not do any consequential changes.

Timing: Let us assume that the original 5-stage pipeline was running at 100 MHz (10ns clock) and we will be running this 3-stage pipeline at 50 MHz (20ns clock). So we did not introduce any timing problems! Since the EX stage was not combined with any other stage, we are planning to add more complex arithmetic operations to ALU to make use of the 20ns clock. So we conclude that the MEM_WB stage cannot help the junior in EX stage but can help the junior in the ID stage through internally forwarding register file.

Briefly comment on the following areas (when space is provided) but you do not have to carry out any changes on the next page. If the unit being commented upon had comparators, state the number of comparators in it.

1. The wrist-band flip-flop and the two inverters associated with it should be removed. [T] [F]
2. The AND gate after the equality checker shall be removed. [T] [F]
3. Wrist-band FF is needed to wrist band the random instruction in the IF_ID stage when you switch on power to the 3-stage pipeline. [T] [F]
4. Like before, clearing the stage registers IF_ID/EX and EX/MEM_WB on reset, makes sure that on power-on, we have bubbles (i.e. no random instructions) in stages EX and MEM_WB.
5. The control unit continues to produce 9 control signals. [Yes] / [No]
6. Internally forwarding in the register file and the number of comparators in it: [Yes] in I.F.R.F.
7. Successful branch, flushing, wrist-band FF, delay slot:
   Since IF and ID are combined into one stage IF_ID, there is no FF after it to flush a branch.  AND FF or to delay a delay slot.
8. No change. After 1 stall, the lw helps the dependent in ID through the I.E.R.F.
9. HDU_Br and STALL_BEQ, and the number of comparators in HDU_Br:
   HDU_Br is simplified to stall beg in ID if there is a Register Writing Instruction in EX stage (lw or r-type). An LW in MEM WB stage helps through IFRF.
10. FU_Br and the two forwarding muxes, and the number of comparators in FU_Br:
    FU_Br and the two forwarding muxes go away as any register writing instruction in the MEM_WB stage would already be helping already through the I.E.R.F.
11. FU and the 4 forwarding muxes, and the number of comparators in:
    Gets reduced to HALF. One pair of muxes are removed (preferably in one receiving FU-RS_WB & FU-RT_WB). 2 comparator units instead.
12. Though (Though / Since) number of stalls are less in the 3-stage pipeline, its overall performance is lower (higher / lower). Because you are running at half the frequency, the few clock cycles you save are insignificant.
No grading is needed for this page!

Early Branch -- 5-stage to 3-stage conversion exercise
Consequential changes are shown here (not asked in the exam)

Control signal portion cleared on reset to ensure bubble are present during RESET

Early Branch -- 5-stage to 3-stage conversion exercise

F0-Br is removed since no Senior #2 in Mem-WB helps his junior in IF_ID Stage through IFRF.

Similarly, half of the FU was removed as there is only one Senior ahead of the EX stage instruction.
(18 + 10 + 8 = 36 points) 15 min. Cache and Main Memory Organization:

A 64-bit data (D63-D0) 32-bit (logical) address byte-addressable processor (address pins: A31-A3, /BE7-/BE0) has its cache and MM organized as shown below. Fill-in the 9 boxes.

5.1 Block size (based on degree of lower-order interleaving of the MM) = ______ Words = ______ Bytes

If set-associative, degree of set associativity = ______ blocks/set

The processor address space = ______ GBytes. Cache size = ______ KBytes

5.2 Please divide the address below into appropriate fields and name the fields.

It is not difficult to get an A in EE457. You need to work for it and seek help from the 457 teaching team on whatever you do not understand. We are eager to help you.

The final topics, virtual memory, exceptions, branch prediction, out-of-order execution, chip multi-threading, chip multiprocessors, cache coherency, locks, and mutual exclusion are interesting and challenging too. They are the focus of 50% of the final exam. Best wishes!