EE457 MT (~20%)  
Closed-book Closed-notes Exam; No cheat sheets; Calcuotators 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

---

**SOLUTION**

Student’s Last Name: ________________________________

Student’s First Name: ________________________________

Student’s DEN Bb username: ____________________________@usc.edu

---

<table>
<thead>
<tr>
<th>Ques#</th>
<th>Topic</th>
<th>Page#</th>
<th>Time</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2nd edition multi-cycle CPU</td>
<td>2-4</td>
<td>40 min.</td>
<td>48</td>
</tr>
<tr>
<td>2</td>
<td>Lab 7 Part 1 3-elem. adder pipeline</td>
<td>5-7</td>
<td>50 min.</td>
<td>69</td>
</tr>
<tr>
<td>3</td>
<td>5-stage Early branch and Branch Delay slot</td>
<td>8-9</td>
<td>50 min.</td>
<td>59</td>
</tr>
<tr>
<td>4</td>
<td>convert 5-stage early branch to 3-stage</td>
<td>10-11</td>
<td>20 min.</td>
<td>35</td>
</tr>
<tr>
<td>5</td>
<td>Cache (reverse engineering)</td>
<td>12</td>
<td>15 min.</td>
<td>36</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>Cover +11</strong></td>
<td><strong>175 min.</strong></td>
<td><strong>247</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Perfect Score**

---

Viterbi School of Engineering  
University of Southern California
1 (48 points) 40 min.

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 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 "PFCJ = 1" or "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?

In State 4 (last state of lw) or State 7 (last state of R-Type) fetch the next instruction at the already incremented PC, increment PC by 4 and proceed to State 1. That is exactly what we do anyways in State 0. But if it is a jump instruction originally we were replacing the PC with the JA (jump address) in State 9 and then we go to State 1 to fetch next instr in State 0 which we are incrementing to PC. Now, using the new mux, not only we are able to fetch the instr at JA in State 9, but also increment that JA.

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 [states 5] (last clock of SW), the memory is busy storing the word with T/D = 1.
Hence prefetching is not possible.
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?

Basically, in the first edition design, in the absence of MDR, memory continues to read the DATA in State 4, and is not available to read the next instruction.
Similarly, in the absence of an ALU output register, it 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 inequality check for beg.
Complete connections to the new mux. Make other needed adjustments.

Original solution with an error in the PC[31:28] portion -- error detected by Fall 2015 student, Shixian Wen <shixianw@usc.edu>. Thanks Shixian for your sharp analysis!

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.
2  ( 69 points) 50 min. 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. \[ \text{T / F} \]

Is this the same in Lab 6 5-stage pipeline discussed in the book/class? \[ \text{Y / N} \]

Reason for similarity or difference: ______________________________________________________________________________________

Stalling: Spurious stalls \[ \text{lower performance} \] (lower performance / produce wrong results).

Forwarding: You care to check if the senior providing forwarding help is not a NOP. \[ \text{T / F} \]

You care to check if the junior receiving forwarding help is not a NOP. \[ \text{T / F} \]

Is one of the above very important? Explain ______________________________________________________________________________________

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? \[ \text{No / Yes} \]

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

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? ______________________________________________________________________________________

____________________________________________________________________________________

____________________________________________________________________________________

____________________________________________________________________________________

____________________________________________________________________________________

____________________________________________________________________________________
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 Part 1 with a dummy stage
All forward controls (X-FORW1, etc.) are produced when the receiving instruction reaches its stage, where the forwarding MUX is located.

Dependency on SI in EX1 and/or dependency on S2 in EX2 leads to stalling.

Forward WB_RD is dependent on the instr. in WB

EX1_ID_XMEM3

X_FORW1

WB_WRITE

EX1_ID_YMEM3

Y_FORW1

WB_WRITE

EX1_ID_ZMEM3

Z_FORW1

WB_WRITE

EX2_ID_ZMEM2

Z_FORW2

WB_WRITE

EX3_ID_ZMEM1

Z_FORW3

WB_WRITE
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 FF or FFs, either here or on the block diagram.

Notice that we are using XOUT signals, so as to make sure that the user has not turned himself into a NOP!

20 pts
3 ( 49 points) 50 min. 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 J etc. executing from the ID stage.

3.2 IF.Flush is crossed off because **we do not want to flush the IF stage instruction.**
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?

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

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.

Explain: Two delay slots => no flushing of IF1 and IF2. However, you need the FF with VDD input so that on Reset, you have bubble-in (P2) and IF2.

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. T / F

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.

March 27, 2015 1:15 am EE457 Midterm - Spring 2015 8 / 12 © Copyright 2015 Gandhi Puvvada
3.2 IF. Flush is crossed off because we do not want to flush the IF stage instruction (because we are supporting a 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?

VDD to GND requires removal of the inverter.

If we remove the VDD connection and restore the crossed-off part, what happens to the
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.
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 can not 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 it's I.E.F.E.
does not change. 2 comparison units (same as before).
7. Successful branch, flushing, wrist-band FF, delay slot: Since I and ID are combined into one stage IF_ID, there aren't after it to flush using a jump. AND FF on to demodulate a delay slot. No change. After a stall, this will help the dependent in ID through the I.E.F.E.
8. HDU and STALL_LW, and the number of comparators in HDU:
9. HDU_Br and STALL_BEQ, and the number of comparators in HDU_Br: comparators reduced to 2 from 4
10. FU_Br and the two forwarding muxes, and the number of comparators in HDU_Br: FU_Br and two forwarding muxes are removed as any rewriting instruction in the MEM_WB stage would be helping already through the I.E.F.E.
11. FU and the 4 forwarding muxes, and the number of comparators in :

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 clocks you save are insignificant.
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

FO-Br is removed since the Senior #2 in Mem-WB helps the 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.
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) = \[ \text{4 Words} = \text{32 Bytes} \]

If set-associative, degree of set associativity = \[ \text{3 blocks/set} \]

The processor address space = \[ \text{4GB (96KB KBytes)} \]

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