Pipelining (Lab 7 Part 1)

Consider the following sequence of instructions run on lab 7 part 1. $9$ is trying to accumulate the sum of the 7 registers $1$ to $7$. Assume that neither EX1\_COUT nor EX2\_COUT will go active

\[
\begin{align*}
&(SR) \leftarrow (Z) + (Y) + (X) \\
&((S9)) \leftarrow (1) + (2) + (3); \text{ -- instruction I} \\
&(S9) \leftarrow (S9) + (4) + (5); \text{ -- instruction II} \\
&(S9) \leftarrow (S9) + (6) + (7); \text{ -- instruction III}
\end{align*}
\]

How many stalls are caused because of the dependency problem? $0 / 1 / 2 / 3$. Explain. ________________________________________________________________________________________

Who receives help through Z1\_mux? instruction(s) occur really (in appears to be receiving) ________________________________________________________________________________________

Who receives help through Z2\_mux? instruction(s) ________________________________________________________________________________________

Who receives (or appears to receive) help through Z1\_mux followed by Z2\_mux? instruction(s) ________________________________________________________________________________________

Z1\_mux and Z2\_mux are (are/aren’t) in natural order and explicit priority is (necessary/unnecessary). ________________________________________________________________________________________

Do you prevent one (or both) of these two muxes from offering inappropriate help? Yes / No ________________________________________________________________________________________

Briefly explain the issues here. ________________________________________________________________________________________

1.2

For the VLSI layout convenience, your VLSI engineer asked you (the architect) to please move 2 out of the 6 comparison units in the Comp Station in ID stage to EX1 or EX2 stage without incurring sizable penalty in terms of speed of the system or cost of the system (cost in terms of silicon area). The 6 comparators were grouped into three pairs in the block diagram on page 3 [(1A, 1B), (2A, 2B), (3A, 3B)]. Comment on the movability of each pair using words such as "impossible", "possible but undesirables", "possible and desirable". Pick the best choice among 6 choices (of moving any of the three pairs to EX1 or EX2) and show the move on page #3. If appropriate, change input and/or output connections and labels and if the move affected the logic of any of the 5 control signals (STALL, X\_FORW1, Y\_FORW1, Z\_FORW1, Z\_FORW2), then show the new logic on page 3 in the rectangle.
Explain your above answers briefly. Choice (i) is impossible because we need to stall the instruction in the ID stage based on the comparisons. Choice (ii) is impossible because the dependent instruction needs to receive help in EX1, and moving the comparator to EX2 means producing the information after it is needed. Choice (vi) is actually partially impossible, and partially possible but undesirable. Partially impossible since the comparator 3B is to be moved to EX2, as by the time the client goes to EX2, the same instruction would have left the pipeline. For the comparator 3A, if we move the comparator to EX2, we will be comparing and will be final, and there will be no critical path. So comparator delay adds to the critical path.

Stalling is currently in the ID stage. Your boss wanted you to move the stall to the EX1 stage and she told you something like, "...we can have a stage after the WB called WB_after...". You do not know if your boss is a Bruin or a Trojan. Discuss the feasibility. If it is feasible, discuss the details of the new design including what goes into WB_after, how many comparators are involved in stalling/forwarding, any changes to forwarding besides stalling, any changes to internally forwarding nature of the RF file to avoid duplication of hardware, overall whether it is desirable or undesirable to do this move. If it is not feasible, state reasons.

For this question, let us keep all comparators in the Comp stage in the ID Stage only.

The question (Q#1.3) was revised (corrected) in Spring 2018.

Please see the revised question and solution at


Produce those of the 5 signals whose logic was affected by your moving of two
comparators to EX1/EX2 stage (STALL, X_FORW1, Y_FORW1, Z_FORW1, Z_FORW2).

EX1_ZWB -> Z_FORW1
WB_WRITE -> Z_FORW2
EX2_EX1_ZMEX2 -> Z_FORW2
WB_WRITE -> Z_FORW2

Basically we are making two comparisons one clock after in EX1

Pipelined 3-element Adder
Block Diagram
LAB 7 Part 1

Fig. 1
1.4 Convert the Lab 7 part 1 into a single-cycle design. Complete the design below including the gate level design of the CU (the Control Unit).

The single-cycle CU is a **combinational** (combinational/sequential) logic.

1.5 Now convert the Lab 7 Part 1 into a multi-cycle design. Here, we use one adder to do \(X+Y\) and further add \(Z\) later. PC has a dedicated incrementer. The register file has only 2 ports instead of 3 ports. There is no IR. Complete the data path below and complete the state diagram for the control unit on the next page. Here, follow the 1st edition design (static logic, long cascading of combinational logic across clocks, etc.). Since there is no IR (instruction register) here, you propose to increment the PC in the **last** (first/last) clock of an instruction execution.

---

The control signal, \(\text{Tmp\_write}\), **can't** (can/can't) be permanently enabled because **in the last 2 clocks**, **in the clock adding **\(Z\), and **in the clock writing back to \(W8)\), the intermediate sum \((X+Y)\) in \(\text{Tmp\_reg}\) should be held constant.

State what modifications to the DPU would have made you answer the other way around to the above question. **If we take the output of \(\text{Tmp\_reg}\) as the RD input of the \(\text{Reg\_file}\), then \(\text{Tmp\_reg}\) will hold \((X+Y)\) when the adder is adding \(Z\) and in the next clock it will hold \((X+Y+Z)\) for deposition into**
Here, we have listed all 6 control signals in all 5 states. For each of the control signals in each of the states, write a 0 / 1 / X, giving preference to X over a 0 or a 1. Draw state transition arrows and write state transition conditions. Consider RUN and COUT signals.

1.5.1 In the above DPU (data path unit without IR) on the previous page, we want to know if it is possible to use the main adder (which adds X+Y, and further adds Z) to do PC incrementation also without increasing the number of clocks. Of course there will be additional muxes to bring PC and the constant 1 to the input of the adder and there will be a connection from the output of the adder to the input of the PC register. It is __________________________ (not possible / possible / possible with an additional important modification). Explain.

_______________________________________________________________________________
_______________________________________________________________________________
_______________________________________________________________________________

1.5.2 Now ignore any modifications suggested by Q 1.5.1 above. Let us now add an IR (Instruction Register) as shown on the side to the original design (which has a dedicated incremernter for the PC). Your boss, a senior EE560 Trojan, tells you that you can fetch the next instruction as you finish the current instruction there by saving a clock! You can do a mealy-type design activating certain signals conditionally.

On the next page, an empty state diagram is given. Record the values of PC_EN and IR_write in each state, draw state transition arrows and conditions. For the rest of the signals, mention them only if you want to (and need to) change them from what they were in the previous state diagram on the top of this page.
1.5.3 If the first 10 instructions are all NOP instructions (with RUN = 0), then they take _______ clocks on the original design on page 4 whereas they take _______ with the design with overlapped instruction fetch in Q 1.5.2

1.5.4 Now ignore any modifications suggested in Q1.5.1 and Q 1.5.2. No more overlapping of fetching of the next instruction in the finishing clock of the current instruction. We want to modify the original datapath on page 4 as per the 2nd edition design (dynamic logic, no combinational cascading across clocks, etc.). Your lab partner added IR register with IR_write control signal, XD register to hold the X register data with XD_write and performed needed connections to these two items.

1. Did he miss to fix the connection to the RD (Result Data) input on the Register file? ____Yes/No Explain and also comment on Tmp_write ____________________________

2. Can XD_write be activated permanently? _______ (Yes / No). If "no", what is the reason. If "yes" state any consequential changes to the DPU and/or CU? ____________________________

3. Can IR_write be activated permanently? _______ (Yes / No). If "no", what is the reason. If "yes" state any consequential actions in the DPU and/or CU? ____________________________

4. Comment on the IFRF aspect (internally forwarding aspect) of the register file in the original design resembling 1st edition design and in this design resembling the 2nd edition design. Consider using words like necessary/unnecessary/harmful/unharmful/useful/useless. ____________________________

_______________________________________________________________________________

_______________________________________________________________________________

_______________________________________________________________________________

Notice that its PC is preparing to increment conditionally in ID, EX1, EX2 stages. And if the instruction is being aborted at this state, we can now go to the ID stage because we are prefetching the next and dropping off in IR.

1.5.3 If the first 10 instructions are all NOP instructions (with RUN = 0), then they take 20 clocks on the original design on page 4 whereas they take 11 with the design with overlapped instruction fetch in Q 1.5.2

4 pts

16 pts

Now ignore any modifications suggested in Q1.5.1 and Q 1.5.2. No more overlapping of fetching of the next instruction in the finishing clock of the current instruction. We want to modify the original datapath on page 4 as per the 2nd edition design (dynamic logic, no combinational cascading across clocks, etc.). Your lab partner added IR register with IR_write control signal, XD register to hold the X register data with XD_write and performed needed connections to these two items.

1. Did he miss to fix the connection to the RD (Result Data) input on the Register file? Yes/No Explain and also comment on Tmp_write _________________________________________

2. Can XD_write be activated permanently? Yes/No. If "no", what is the reason. If "yes" state any consequential changes to the DPU and/or CU? ____________________________

3. Can IR_write be activated permanently? Yes/No. If "no", what is the reason. If "yes" state any consequential actions in the DPU and/or CU? ____________________________

4. Comment on the IFRF aspect (internally forwarding aspect) of the register file in the original design resembling 1st edition design and in this design resembling the 2nd edition design. Consider using words like necessary/unnecessary/harmful/unharmful/useful/useless. ____________________________

_______________________________________________________________________________

_______________________________________________________________________________

_______________________________________________________________________________

Because of the YD reg and temp reg, there is no harm due to IFRF. It is superficial but harmless in the 2nd Ed design.
Datapath design similar to the multicycle CPU from the 2nd edition
Cache mapping techniques:

Fill-in all missing information in the table below based on information provided. In all four cases, it is the same amount of cache differently organized.

<table>
<thead>
<tr>
<th>Addr Space Size</th>
<th>Cache Size</th>
<th>Block Size</th>
<th>Mapping Technique</th>
<th>TAG FIELD</th>
<th>BLOCK OR SET FIELD</th>
<th>WORD FIELD</th>
<th>BYTE FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 2-bytes</td>
<td>8 KB</td>
<td>256 Bytes</td>
<td>Direct</td>
<td>A31-A12</td>
<td>19-bit</td>
<td>A4-A3</td>
<td>A2-A0</td>
</tr>
<tr>
<td>4 2-bytes</td>
<td>8 KB</td>
<td>256 Bytes</td>
<td>Fully Associative</td>
<td>A31-A12</td>
<td>19-bit</td>
<td>A4-A3</td>
<td>A2-A0</td>
</tr>
<tr>
<td>4 2-bytes</td>
<td>8 KB</td>
<td>256 Bytes</td>
<td>Set Associative 2</td>
<td>A31-A12</td>
<td>19-bit</td>
<td>A4-A3</td>
<td>A2-A0</td>
</tr>
<tr>
<td>4 2-bytes</td>
<td>8 KB</td>
<td>256 Bytes</td>
<td>Set Associative 8</td>
<td>A31-A12</td>
<td>19-bit</td>
<td>A4-A3</td>
<td>A2-A0</td>
</tr>
</tbody>
</table>

TAG RAM(s) and their size(s) and comparators to compare TAG(s) and their size.

In the case of Direct Mapping above, we use 1 (state a number) TAG RAM(s) of size 256 x 20 (as appropriate) together with 1 (state a number) comparator(s) each of 19-bit wide.

In the case of Set Associative Mapping with 2 Blocks/Set above, we use 2 (state a number) TAG RAM(s) of size 128 x 21 together with 2 (state a number) comparator(s) each of 20-bit wide.

In the case of Set Associative Mapping with 8 Blocks/Set above, we use 8 (state a number) TAG RAM(s) of size 32 x 23 together with 8 (state a number) comparator(s) each of 23-bit wide.

The Fully Associative Mapping is prohibitively expensive because you would need 225 locations x 20 bits/or 20 bits include 19-bit tag + 1-bit valid.

In the first case of direct mapping, the main memory shall be organized in a 4-way lower-order interleaving to facilitate efficient access of a block of 4 words.

In general, a set can potentially have a set-associativity equal to any number (not necessarily a power of 2). However, here (However, here / Here also) in the above table, the set associativity needs to be (needs to be / does not need to be) a power of 2 in size, because the cache size (based on the direct mapping info provided in row 1) is already given in the above table.

If the total number of block frames in a cache is N, we can bring into that cache, if it is initially empty, any consecutive N blocks from the main memory (without causing any collision), in the case of cache using the mapping technique (circle all correct answers): all three.

Fully-associative  Set-associative  Direct

March 30, 2012 5:37 am  EE457 Midterm Exam - Spring 2012 Page - 7 / 8
Virtual Memory:

3.1 A 32-entry 2-way set-associative TLB contents (in hex) are given on the side. In this virtual memory system, 64KB pages are used. The processor is a 32-bit logical address 64-bit data byte addressable processor.

1. Divide the Virtual Address into VPN and Page offset fields. Divide the VPN further into Tag and Set fields.

2. To make sure that I did not fill random VPNs, which can potentially make up a meaningless TLB content, I should _____________________________________________________________
   _____________________________________________________________

   Similarly, to make sure that I did not fill random PPFNs, which can potentially make up a meaningless TLB content, I should ___________________________________________________
   _____________________________________________________________

3. If the TLB is all empty to start with, translation for how many consecutive virtual pages we can hold in the TLB? _______. If we are unlucky, how early can we get conflict, requiring replacement in TLB? _____________________________________________________________

4. Based on the TLB contents, arrive at the following:
   Number of valid entries in the TLB _______. List all those valid VPNs for which we have translation here: _________________________________________________________________
   Translations for two consecutive Virtual pages _______ (always / never / sometimes) go into the two entries of a set.
   Are there any consecutive (valid) VPNs? _____ (Y / N). If Yes, list them here: _______

5. How many comparators of what size are used in this TLB? ______________________________
   _____________________________________________________________

3.2 Since we use "write-only" for virtual memory, we need to maintain a dirty bit associated with each entry in both TLB and page table (the entire TLB /the entire Page Table / each entry of the page table / each entry of the TLB / each entry in both TLB and Page Table).

3.3 Fully associative mapping may not be prohibitively expensive in the case of a _______ (TLB / L2 cache) because _______.

The next few weeks are very important as we will be covering a lot of material in 4 weeks. Please, do attend every lecture and discussion. And use our office hours. Thanks. -- The EE457 Teaching Team.