Pipelining

The MIPs 5-stage late branch pipeline design is reproduced on the next page. As you see, the branch is executed from the MEM stage and a successful branch flushes the next three instructions. The BR1 signal also bears the three names EX_Flush, ID_Flush, and IF_Flush. Note: The BR1 simply enters the control unit and comes out with three new names.

For the sake of this question, assume that the ID stage is having timing problems and that the instruction decoder and the 9-bit 2-to-1 bubble-injecting mux are in the critical path.

Your boss, Miss Trojan proposed that we remove the bubble-injecting mux from the ID stage and add a "Bubble Bit" (similar to "flush bit", basically a new "wrist-band") to the instruction going into the EX stage whenever the instruction in ID stage is stalled or being flushed or "destined to be flushed". To simplify (thereby speedup) the instruction decoder, she suggested that the incoming instruction, "destined to be flushed", is made to carry a "Bubble Bit" into the EX stage so that it gets converted to a bubble in EX stage.

1.1 Like in Lab 7 Part 3 Sub-part 3, to transfer a mux into the next stage, we expect to carry the two sets of 9-bit inputs to the mux into the next stage. But let us examine the current context and see what is necessary.

Do you want to carry 9 zeros (a constant) into the next stage? Yes / No

We noted that it was not really necessary to convert all 9 control signals into zeros, only a significant subset are important, namely ____________________________________________

We further note that there are two flush multiplexers in the EX stage controlled by EX_Flush. May be we can make use of them instead of adding a new 9-bit flush multiplexer in the EX stage.

So, until it is flushed by the EX_Flush muxes, this instruction in EX stage may be mistaken to be real. What if it is a lw (load word) instruction and the HDU in ID stage stalls spuriously? Is it the question of losing a clock or is it a tug of war between a successful branch and the HDU? Who should win? If it is a tug of war, did we create it now because of our new modification or was it existing and the design on the next page needs to be corrected? Explain: __________________
____________________________________________________________________________
Original (OLD) Lab 6 late branch block diagram - Answer Q 1.4.1 here
1.2 Reset actions:

Notice that I have added \( R \) (RESET) signal to the PC to clear it on reset on the next page. It was implicit in the block diagram on previous page. In the diagram on the next page, make all reset actions explicit. On reset, you want to ____________________ (clear / preset / leave it without reset action) the rest of the 4 stage registers (IF/ID, ID/EX, EX/MEM, MEM/WB). Reason: ____________________________

The original "flush bit" is active-low (active-high/active-low) and hence it is ____________ (cleared/preset) on reset. You are planning to make the new "Bubble Bit" FF (being added by you to the ID/EX stage register) active-low (active-high/active-low). Your lab partner says that it is unimportant to set or reset this Bubble Bit on reset. You __________ (agree / disagree). Explain: ________________________________________

1.3 Show your design on the next page.

Are we losing or gaining any clocks because of removing flush mux from ID stage? ____________

1.4 If the ISA defines one branch delay slot, what would you do under each of these two cases:

(1.4.1) In the original late branch design without the current modifications (consider also the tug-of-war cited before). Show your design on page 2. Briefly explain: ____________

(1.4.2) In the design with the current modifications (consider also the tug-of-war cited before). Show your design on page 5. Briefly explain: ____________________________

1.5 Now, instead of the late branch design, let us think of the early branch design with no delay slot.

In this case the ID stage ______________ (does / does not) get flushed because of a successful branch. ____________ (and/or) it ______________ (does / does not) get stalled due to ______________ (RAW/WAW/WAR) hazards. The ID stage ______________ (does / does not) receive a flush bit. Hence there is a ____________ (is / isn’t) any possibility of injecting a bubble into EX stage for timing reasons.

Unlike ____________ (Like / Unlike) in the late branch, there weren’t any (were two / weren’t any) flush muxes in the EX stage. Hence, we ______________ (use those two muxes in the / add two muxes to the) EX stage and place them under the control of the "Bubble Bit".

State what is the impact of a delay slot, on what you said above. ____________________________

More than 1 delay slot can be thought of in ____________ (early branch only / late branch / both/neither). Because ______________ (there are 3 junior instructions behind a late branch but...
Show your design for Q 1.4.2 (with 1 branch delay slot) her.

In this design the Bubble Bit is active high.
Solution to a question from your midterm is reproduced below. Solution lines are dotted.

Question: 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).

Now we modify the basic instruction from \( R = X + Y + Z \) to \( R = X + Y - Z \). We add \( X \) and \( Y \) and from the sum we subtract \( Z \). Like in the original problem, all are unsigned 16-bit numbers. If the overall operation \( (X+Y-Z) \) produces overflow (either by being larger than what can fit in the 16-bit number \( R \) or by being negative), then we do not write the result in the result register \( R \). Note that if \( (X+Y) \) produces \( COUT_1 \), you should not conclude that there is overflow because the sum may have gone beyond the limit temporarily and after subtraction of \( Z \), it may come back to within limits. Complete the design on the next page.

To subtract \( Z \), you can add \((Z'+1)\). In the diagram below, we have shown \( Cin \) (carry in) so that you can tie it to VCC or GND as needed. Perhaps it is easier to detect overall overflow if we are given a 17-bit adder and a 17-bit subtractor to operate on the 16-bit operands, \( X, Y, Z \). Keep the two adders as 16-bit adders and add external logic to produce overall overflow.

Overflow in unsigned subtraction means the result is negative and is unrepresentable in unsigned system.

If we use a 17-bit adder to add two 16-bit numbers to produce a 17-bit result, there will never be an overflow whether we are performing unsigned addition (all numbers are unsigned) or signed addition (all numbers are signed).

If we use a 17-bit subtractor to do A-B (A and B are 16-bit unsigned numbers) to produce a 17-bit unsigned result, there will never be a overflow.

Think of the four combinations on the side and add a remark letter.

<table>
<thead>
<tr>
<th>COUT_1</th>
<th>COUT_2</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>D</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>E</td>
</tr>
</tbody>
</table>
Complete the design below.  \( \text{Cout} - 1 = 1 \Rightarrow \text{a value of} \ 2^{16} \). \( \text{Cout} - 2 = 0 \Rightarrow \text{indicates need for help from that } 2^{16} \text{ to complete subtraction.} \)

2.2 Now, we are asking you to do \( R \leq X + Y - Z \) where \( R, X, Y, Z \) are all 16-bit signed numbers represented in 2’s complement notation. We are giving you two additional full-adders so that you can convert the 16-bit adders to 17-bit adders if it helps you. Again if the result does not fit into the 16-bit \( R \), you should not store the result. You are responsible to deal with any sign extension overflow detection, etc. Use \( X[15] \) to represent left-most bit of \( X \) and similar notation for others.

We can safely drop the left-most bit of a 17-bit signed 2’s complement number and store the lower 16 bits, if \( RD[16] \text{ and } RD[15] \) are identical.

My design is wrong. See next page.
Mr. Chen Qian's comment:
problem 2.2, I think it should be R[17] R[16] R[15] should be identical, and R[17] is V^R[16], I think that we should look one more bit for x+y-z operation.

I agree with Chen. My design on the previous page is wrong because three signed 16-bit items can not necessarily fit into a 17-bit item. To produce positive most result R, we need to use largest positive values for X and Y and largest negative value for Z. Similarly if X and Y are largest negative values and Z is largest positive value, we produce negative-most result.

Let us take 4-bit numbers instead of 16-bit numbers.

Largest positive result R = 0111 + 0111 - 1000 = +7 + 7 - (-8) = +22
We need a 6-bit number to say +22

Largest negative result R = 1000 + 1000 - 0111 = -8 -8 -(+7) = -23
We need a 6-bit number to say -23

We need the extra one more bit (18th bit) when we make the subtraction of Z only (and not when we do the addition X + Y) because X+Y can fit in 17 bits. Instead of using an 18-bit subtractor, we can infer what is the R17 from V and R16. From our SLT implementation we know that R17 = V XOR R16 as Chen said in his email.

Implementation of logic to check to see if R17, R16, and R15 are all the same:
Here I am separately checking to see if R16 and R15 are the same and R17 and R16 are the same. If both pairs are the same, then all three bits are the same. If the V bit in the 17-bit subtractor is true then R17 and R16 differ. Logical reasoning: If the 17-bit subtraction could produce a correct result in 17 bits, then there will not be any overflow (V =0). Such 17-bit result can be written as an 18-bit result by sign-extending, which means R16 and R17 are the same. On the otherhand if V = 1, it means that the 17-bit result is wrong and a supposedly large positive result is appearing as negative (or a supposedly large negative result is appearing as positive). It means an 18-bit result when truncated to 17 bits is coming out to be bad. So R17 and R16 differ.
Further, if the result did not fit in 17 bits, it will certainly not fit in 16 bits.

Conclusion: If the 17-bit subtraction did not result in an overflow (V =0) (which means that the result could fit in 17 bits), and further if we find that R16 and R15 are identical (meaning the result can actually fit in 16 bits) then we can write the result in the Register File.

Thank you Chen!
Date: 12/12/2012

Another interesting implementation for the signed number case:

Let us assume that the C[15]_1 and C[15]_2 are accessible besides C[16]_1 and C[16]_2. Then we can do better. We can find when one error cancels another error. This is similar to the approach we took for the unsigned numbers. Having access to C[15]_1 and C[15]_2 provides more information than having access to just the XOR of the two items.

Classnotes reference: See slide 22/56 and related slides of the Chapter 4 handout to convince yourself of the loss or gain of 2**16.

http://www.classes.usc.edu/engr/ee-s/457/EE457_Classnotes/EE457_Chapter4/ee457_twos_comp1_RCA_RBS.pdf

I believe that the 4-input combinational logic to produce the overall overflow for the adder + subtractor cascade will be cheaper and faster compared to the two additional full adders followed by some additional logic on the previous page.

A non-SOP implementation of the combinational logic is as follows. Consider that the highlighted squares are initially filled with 1's. Then the tentative solution is that the overall overflow is true if there is any overflow in any one of the two items (adder or subtractor). Then fix the tentative solution by removing the two cases as shown below.
Date: 12/12/2012 (with improvements suggested Jintao Shen)

Another interesting implementation for the signed number case:

Let us initially assume that the C[15]_1 and C[15]_2 are accessible besides C[16]_1 and C[16]_2.

This helps in our initial analysis though finally we note that they are not needed.

We wish to find when one error cancels another error.

This is similar to the approach we took for the unsigned numbers.

---

Classnotes reference: See slide 22/56 and related slides of the Chapter 4 handout to convince yourself of the loss or gain of 2**16.
http://www-classes.usc.edu/engr/ee-s/457/EE457_Classnotes/EE457_Chapter4/ee457_twos_comp_L_RCA_RBS.pdf

Mr. Jintao Shen (EE457 Fall 2012) noted that the row #5 and row #10 in the above truth table are impossible as we can not lose -2**16 two times in row #5 and we can not lose +2**16 two times in row #10 from any combination of input three numbers X, Y, Z. So output produced for these impossible input combinations can be any convenient value. Jintao proposed a "0" output for these rows so that a simple XOR of the two individual overflows becomes the overall overflow!

Good observation, Jintao!!!
For your information, the incomplete block diagrams of both Lab 7 Part 1 & Part 2 are given on the next page. You coded part 1 in structural Verilog and completed part 2 on paper.

The structural code of part 1 consisted of `assign` statements and `instantiations` in a readable order. For example, we instantiated `ID/EX1` stage register and then wrote a bunch of `assign` statements to describe the logic in the `EX1` stage and then instantiated the `EX1/EX2` stage register.

How come we did not care to code WB stage first and then EX2, followed by EX1 like in the RTL coding of Lab 7 Part 3 Subpart 3? What is the difference? The `assign` statements are concurrent and describe the combinational logic between the registers. Of course all instantiations are concurrent. So in conclusion, the coding of Lab 7 Part 1 is by writing a bunch of `assign` statements and `instantiations` which are all concurrent and can be written in any order.

Conclusion: The `assign` statements and the `instantiations` in a structural code can be in any order. T/F

Writing many `assign` statements in a Verilog code **is** (is / isn’t) like a data-flow description and **isn’t** (is / isn’t) discouraged. Encouraged.

It is **possible** (possible / not possible) to replace all (not just one, all) the `assign` statements in your Lab 7 Part 1 with one `always` procedural block. If it is possible, it will be a **combinational** (combinational / clocked / either) block. If not possible, the reasons are ___________________________.

Irrespective of whether all or a subset of `assign` statements could be replaced by an `always` procedural block, do you think you would mostly be using `blocking assignments` or `non-blocking assignments` inside this always procedural block? **blocking** (blocking / non-blocking). Explain: **In coding deep combinational logic, we want to produce some intermediate results and using them, we want to produce next intermediate results and so on. Hence all these intermediate results are produced with an intent to consume immediately and hence they should be produced using `blocking assignments`. The final results of a combinational logic can be produced using `blocking` or `non-blocking assignments`. So a thumb rule to follow is use `blocking` for combinational logic.**

Suppose you are coding the parts 1 and 2 using the RTL coding methodology. The `WB_RD`, `WB_RA`, and `WB_WRITE` (from the `WB` stage in part 1 and from the `EX2WB` stage in part 2) are used in the Register file. It is proposed that the RTL consists of **one or more clocked blocks** only and **no separate combinational logic blocks**.

Part 1 can be coded in one clocked block. T/F

Part 2 can be coded in one clocked block. T/F

In **part 1 only** (part 1 only / part 2 only / both parts / neither part), we can code the `write` port of the register file (which logically belongs to the `WB` stage) in a separate clocked block as shown on the side.
1. Complete all missing connections marked in dotted lines. Also complete the RA (Result Address) connection in ID stage (ID_RA).
2. Complete all five enable (EN) controls on the pipeline registers (including PC).
3. Complete the forwarding paths into the four forwarding muxes.
4. Complete logic to inject bubble into the next stage if the current instruction is being stalled or being flushed.
5. Draw the logic on a separate page for generating STALL, X_FORW1, Y_FORW1, Z_FORW1, Z_FORW2.

Pinelined 3-element Adder Block Diagram
LAB 7 Part 1

Fig. 1

This figure is basically the same as the Fig. 1 for part 1 on page 2 except that the EX2/WB stage register was removed as we are merging the EX2 and WB stages into one EX2WB stage.

You must remove some items which are no longer necessary and complete the rest.
You need to generate any STALL, X_FORW, or Y_FORW signals on this page itself.

Pinelined 3-element Adder Block Diagram
LAB 7 Part 2 (with EX2 and WB merged)

Fig. 2
4.1 WP (the write pointer) in the Tag Fifo in the left design is used to \(\text{put back a "token" into}\) (draw a "token" from / put back a "token" into) the Fifo. WP in the ROB in the right design is used \(\text{to allocate a ROB slot to the next inst being dispatched}\) (to commit the senior-most instruction in ROB, to allocate a ROB slot to the next instruction being dispatched).

4.2 Virtual queue is formed in the \(\boxed{\text{right-side}}\) (left-side / right-side) design because of \(\text{Tag from Tag FIFO / ROB Tag}\). In a store, after pulling a number, we can stand anywhere, and hence we are said to be in a virtual queue. How is the phrase "virtual queue" applicable here? \(\text{Instructions "stand" as per their seniority in the ROB, but they can be in any of the 4 instruction queues and may appear on CDB with no specific order and arrive in ROB from CDB out of order, but they still go and sit in their assigned slot in ROB!}\)

4.3 \(\text{WAW and WAR (WAW/WAR/RAW if multiple of these, list all of those) problems in registers in the right-side design by strict in-order commitment (dispatch/issue/commitment). WAW for sw (store-word) instructions is solved by memory-disambiguation rules in the \(\boxed{\text{left-side}}\) (left-side/right-side) design where as they are solved in the other because of \(\text{above strict in-order commitment again}\).}\)

4.4 In a design, usually one of the components is too big perhaps because another component is too small. Your teammate decided to double the size of all the instruction queues (the MULT queue, the DIV queue, etc. in front of the functional execution units) when processor speed went up. L2 cache speed did not go up. Explain why. And if it is found that the dispatch unit is found to stall more often after this increase, what else he forgot to increase which made the dispatch unit stall? \(\text{In a design, usually one of the components is too big perhaps because another component is too small. Your teammate decided to double the size of all the instruction queues (the MULT queue, the DIV queue, etc. in front of the functional execution units) when processor speed went up. L2 cache speed did not go up. Explain why. And if it is found that the dispatch unit is found to stall more often after this increase, what else he forgot to increase which made the dispatch unit stall?}\)
Three diverging arrows from the O and E states were erased. Restore them with the conditions.

If a cache has a block in "O" or "M" or "E" states, and another cache requests that block, is that block going to be supplied by the Main Memory or the first cache? ___________.

Out of the five states, we can say the block in our cache is the only copy in states M and E.

Avoiding aliasing in BTB (Branch Target Buffer) is very important if we are predicting from ___ (IF / ID) stage.

Left side is the original direct-mapped 2K deep BTB. Right-side is a 2-way set-associative BTB with two 1K deep parts (BTB0 and BTB1). Your colleague says, "It is less likely to have two branches at different PCs (say 2040H and 6040H) with the same index (40H) which conflicting for the same location in BTB. So a lot of sets in the right design will have one entry left unoccupied. So the left design is better than the right design cost-wise and performance wise." What do you say?

It is common to use a block-size of 4 words in the context of cache to take advantage of the spacial locality. If CPU asks for 4000, the cache control unit brings 4000, 4004, 4008 and 400C. ___________.

In a demand paging system, if there is a page fault for the virtual page 200, the OS ___________. If there a TLB miss, the MMU ___________. A2-level page table compared to a 3-level page table is ___________.

We enjoyed teaching this course. Hope you liked the course. Hope to see some of you in EE454L or EE560. Grades will be out in a week.

Enjoy your summer break! - Gandhi, Jonathan, Varun, Gourav, Naizath, Janam, Zhenghui, Rajdeep, Omkar