Arithmetic:

This is similar to a Spring 2012 problem. We need to find overall overflow in addition/subtraction involving three 4-bit operands X, Y, Z and a 4-bit result R for both unsigned and signed numbers. We know that sometimes two errors can cancel each other. So we will be careful to look for them in arriving at the overall overflow.

Four designs are given below. Cross out any inappropriate designs and label the rest as "adder or subtractor" for "unsigned/signed/both". Basically two designs are given below with Cin connection changed from GND to VDD.

In the case of an unsigned 4-bit adder, if the overall carry output C[4] is a __________ (0 / 1), it indicates an overflow meaning a loss of __________ (2\(^4\) = 16 / 2\(^4\) = 16) in the value represented by the 4-bit result R.

In the case of an unsigned 4-bit subtractor, if the overall raw carry output C[4] is a __________ (0 / 1), it indicates an overflow meaning a gain of __________ (2\(^4\) = 16 / 2\(^4\) = 16) in the value represented by the 4-bit result R.

In the case of a signed 4-bit adder, if the C[3] = 1 and C[4] = 0, it indicates an overflow meaning a loss of +2\(^4\) = +16 (2\(^3\) + 2\(^4\) = 16 / -2\(^3\) + 2\(^4\) = -16) in the value represented by the 4-bit result R.

Six incomplete circuits are given below for producing the following three results. The three circuits on page 2 are for unsigned numbers and the three circuits on page 3 are for signed numbers.

(a) R = (X + Y) - Z
(b) R = (X - Y) - Z
(c) R = X - (Y + Z)

Connect GND or VDD to the Cin (carry in) connections and produce overall overflow carefully considering any error-cancellation opportunity. Provide example decimal numbers for X, Y, Z, and R demonstrating such opportunity if it exists.
Three circuits for **UNSIGNED** Numbers

**a**

\[ R = (X + Y) - Z \]

- There isn't any opportunity for error cancellation. \( \text{T/F} \)
- Example decimal numbers demonstrating such opportunity if it exists:
  \( X = 4, Y = 15, Z = 15, R = 4 \).

**b**

\[ R = (X - Y) - Z \]

- There isn't any opportunity for error cancellation. \( \text{T/F} \)
- Example decimal numbers demonstrating such opportunity if it exists:
  \( X = , Y = , Z = , R = . \)

**c**

\[ R = X - (Y + Z) \]

- There isn't any opportunity for error cancellation. \( \text{T/F} \)
- Example decimal numbers demonstrating such opportunity if it exists:
  \( X = , Y = , Z = , R = . \)
Three circuits for SIGNED Numbers

**Diagram a:**

\[ R = (X + Y) - Z \]

Since two +16s or two -16s can't be lost simultaneously, if there is an overflow in each one of the two adders, the errors must be opposing and hence cancelling.

Example decimal numbers demonstrating such opportunity if it exists:

\[ X = +4, \ Y = +7, \ Z = +7; \ R = +4 \]

Another example:

\[ X = -4, \ Y = -8, \ Z = -8; \ R = -4 \]

**Diagram b:**

\[ R = (X - Y) - Z \]

Since two +16s or two -16s can't be lost simultaneously, if there is an overflow in each one of the two adders, the errors must be opposing and hence cancelling.

Example decimal numbers demonstrating such opportunity if it exists:

\[ X = +1, \ Y = -8, \ Z = +3; R = +6 \]

Another example:

\[ X = -1; \ Y = +8; \ Z = -3; \]

\[ R = -6; \]

**Diagram c:**

\[ R = X - (Y + Z) \]

Since two +16s or two -16s can't be lost simultaneously, if there is an overflow in each one of the two adders, the errors must be opposing and hence cancelling.

Example decimal numbers demonstrating such opportunity if it exists:

\[ X = +7, \ Y = +5, \ Z = +5; \ R = -3 \]

Another example:

\[ X = -7; \ Y = -5; \ Z = -5; \]

\[ R = +3; \]

Note: Except for the carry-in connections, all three overall overflow circuits and the explanations are identical. Students do not have to write explanations.
Topic: pipeline modification

The original 5-stage late branch pipeline is given on the next page for your reference. You need to modify the incomplete design on page 6 to satisfy the following requirements from our military. They wanted you to build a special MIPs processor to confuse the enemy forces, who may steal our binaries and try to reverse engineer.

We know that our ISAs usually have several instructions which are not absolutely necessary but they are there for improving efficiency. Our military engineers have selected one such instruction. The special 5-stage pipelined processor (to be designed by you for them) will not execute that special instruction (with special opcode) as the normal MIPs processor would do. Instead, this instruction goes into EX stage and sits there invalidating the next sequence of instructions arriving in the ID stage until a \textit{beq} instruction (control signal \texttt{BRANCH} = 1) arrives in ID stage. Then this special instruction’s job is over and it leaves the EX stage and goes away as a NOP. Normal execution resumes with the first branch after the special instruction.

1. Assume that the control unit produces \textbf{9 zeros} for the 9 control signals for this special opcode.

2. A successful branch \textit{senior to} this special instruction should be able to flush this special instruction when it is in \textbf{EX} or \textbf{ID} or \textbf{IF} stages.

3. Arrangements shall be made to \textbf{hold} the special instruction in \textbf{EX} stage by perhaps controlling the \texttt{EN} (enable pin) on the \texttt{ID/EX} stage register. This perhaps \textit{nullifies} instructions following it!

4. Since 9 zeros for the 9 control signals arrive in the \textbf{EX} stage, those zeros will keep traveling from \textbf{EX} to \textbf{MEM} and then to \textbf{WB} as long as this special instruction is held in \textbf{EX} stage.

5. When eventually a \textit{beq} instruction (control signal \texttt{BRANCH} = 1) arrives in \textbf{ID} stage, he should not be nullified, but should be allowed to move into \textbf{EX} stage. This means the \texttt{ID/EX} enable control needs to be \textit{reactivated}. Arrival of the branch in \textbf{EX} stage should mean removal of the special instruction from the \textbf{EX} stage.

6. Though unrelated to this design, fix any existing \textit{conflict} between a successful branch in \textbf{MEM} stage and the \texttt{HDU} in the \textbf{ID} stage.

7. A \texttt{DCMT} (Data Cache Miss true) signal is generated in the \textbf{MEM} stage for you to stall the entire pipeline including \texttt{ID/EX} stage register to \textit{stall} as long as \texttt{DCMT} is a \textbf{1}.

8. Note that the special partial modifications shown below are already carried out for you on page 6.
Pipelined CPU (late branch from Ist Ed.) for reference only
Graders: Please make sure that the pink line does not go through the ID/EX Stage.

The special instruction arrived in the ID Stage, so we need to flush it.

If currently, there is a successful branch in MEM Stage and ID/Flush is active, then we need to flush it.

Disabling the ID/Ex Stage in register while keeping PC and IF/ID Stage registers enabled, effectively flushing the instruction in the ID Stage without actively intervening and ensuring its control signals to zero. Since a new instruction is coming into ID Stage, and since the instruction can not go into EX stage, it basically gets erased.

Branch = 1

"O" on Reset is good
Point distribution for grading:

- **20 points**
  - Complete diagram of processor pipeline stages.
  - Correctly labeled control signals.

- **15 points**
  - Correctly labeled signals and data paths.

- **10 points**
  - Correctly labeled registers and memory access.

- **8 points**
  - Correctly labeled branch prediction.

- **5 points**
  - Correctly labeled instruction fetch and decode.

- **3 points**
  - Correctly labeled execute stage.

- **2 points**
  - Correctly labeled memory access and write back.

- **1 point**
  - Correctly labeled interrupt handling.

- **0 points**
  - Errors in labeling, incorrect diagram structure, or missing components.
In the design on previous page, "retaining the FF in the set state" is done through disabling the EN pin. Another way of designing the logic is to treat this single flip-flop as the state-memory of a 2-state state machine and manage all three actions listed below through an appropriate next state logic:
(a) setting the FF, (b) retaining it in set state and (c) finally clearing it.

![State Transition Diagram](image1)

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbolic</td>
<td>Coded</td>
</tr>
<tr>
<td>N. E.</td>
<td>0</td>
</tr>
<tr>
<td>S. L.</td>
<td>1</td>
</tr>
</tbody>
</table>

We will make \( Q = 1 \) if \( C_1 \) cond. is prevailing or if it is already set, we let it remain set as long as \( C_2 \) is not true.

\[
Q^* = Q \cdot \overline{C_2} + C_1
\]
This implementation is shown on the next page. We retained the EN control for DCMT to stall the pipeline.

C1 condition

ID-Flush (Wrist-Band)

Q, \bar{C}_2

can be Redrawn as

Q \cdot \bar{C}_2

BRANCH (from ID Stage)
After the flush (i.e., after max (i.e., max. ID Flush)
Acted on it)}
It will be wrong to tap the branch signal from here as it amounts to doing combinational feedback. EX/MEM flushes its juniors as long as it is not a branch instruction.
**3.1** RP (the write pointer) in the Tag Fifo in the left design is used to **draw a "token" from** the Fifo. RP in the ROB in the right design is used to **commit the senior-most instruction in ROB** to allocate a ROB slot to the next instruction being dispatched.

**3.2** Virtual queue is necessary in the __right-side__ (left-side / right-side) design because (a) need for IoC (in-order commitment) (b) we can commit OoC (out of order commitment). Since both TAG FIFO in the left-side design and the ROB in the right-side design are both FIFOs, Mr. Bruin thinks that both form Virtual Queues whether you need it or not. Educate Mr. Bruin: __________________________________________________________________________________________

**3.3** Because of strict IoC in the __right-side__ (left-side / right-side) design, only __RAW__ (WAW/WAR/RAW, if multiple of these, list all of these) problem(s) for memory locations need(s) to be taken care by forcing a __lw__ (lw/sw) wait until all senior lw/sw instructions with matching address retire (commit). Hence it is necessary (necessary/desirable) to maintain instructions in the LSQ (load-store queue) in the order of arrival (in the order of seniority).

A __lw__ (lw/sw) instruction goes into the data cache before going on to the CDB where as a __sw__ (lw/sw) instruction goes into the data cache at the time of commitment. A __lw__ (lw/sw) instruction in LSQ can not go into the data cache if there is a __sw__ (lw/sw/either) in front of it, who has not calculated his effective memory address yet.

**3.4** In the left-hand side design, the dispatch unit consults __RST__ before reading the register file for the source register(s) needed by the instruction being dispatched, where as in the right-hand side design it consults __ROB__ for the same reason. In the __right-side__ (left-side / right-side) design, this results in an expensive associative search of the __ROB__ for each source register.
3.5  Every register-writing instruction writes into the register file in the \textbf{right-side} (left-side / right-side) design where as that is not the case with the other design. Explain how some register-writing instructions do not get to write into the register file in the other design.

\begin{itemize}
\item 1st add $2, \$2, \$2 \text{  If these instructions are dispatched one after another}
\item 2nd add $2, \$2, \$2 \text{  in the left-side design, \$2 gets renamed 3 times as}
\item 3rd add $2, \$2, \$2 \text{  say DOG, CAT, PIG. So RST will have DOG written}
\end{itemize}

The 2nd add is dispatched. The DOG across \$2 in RST gets overwritten with CAT and then with PIG. So when the 1st add comes on CDB, dispatch unit ignores it as no register in RST is currently waiting on DOG. So 1st and 2nd add never get to write into RF.

3.6  LS-Buffer is used at the exit of the Data cache in the above two designs but there isn’t any such buffer after the integer ALU or the multiplier or divider because the data cache is a variable latency execution unit (taking lot more clocks if there is a cache miss). Hence the issue unit can’t allocate a time slot in future for a new instruction going into the data cache. Hence a LS Buffer is provided after the data cache. Though SW instructions do not go through the data cache here, for simplicity, all LW/SW instructions are asked to gather in the LS Buffer. The issue unit will serve the LS Buffer instead of LS Queue.

The 16x2 BPB (Branch Prediction Buffer) in the right-side design has a READ PORT A and a READ/WRITE PORT B with separate Data-in and separate Data-out as shown on the side. We use 2-bit predictor with a 4-state diagram.

We need \(\text{(i) one incrementer/decrementor for the whole BPB (ii) 16 incrementers/decrementers for the 16 locations of the BPB. The dispatch unit uses the port A (A / B) for predicting the conditional branch being issued and uses the port B (A / B) for incrementing/decrementing the content of the location corresponding to the conditional branch on the CDB. The BPB is more like an array of registers (latches / registers).}

\text{Note: To build a counter } \text{ \(i \pm 1\text{, or } i \pm (-1)\text{, you need } i\text{ to be a register.}}\)

3.7  The ROB is a 32-location FIFO in our EE560 implementation. We used 6-bit pointers for WP and RP to distinguish the FULL and empty conditions. Say the WP and the RP are pointing to the same location as shown in the diagram.

If the FIFO is FULL, what are the possible 6-bit values for the WP and RP?

\begin{itemize}
\item (i) \(WP = 000010\) ; \(RP = 000010\) \(\text{At this point (a) (a)(b).}
\item (ii) \(WP = 000010\) ; \(RP = 100010\) \(\text{At this point (b) (a)(b).}
\end{itemize}

Choices: (a) the dispatch unit should stop dispatching (b) the committing should stop

If the FIFO is EMPTY, what are the possible 6-bit values for the WP and RP?

\begin{itemize}
\item (i) \(WP = \text{empty} \) ; \(RP = \text{empty} \) \(\text{At this point (a) (a)(b).}
\item (ii) \(WP = 100010\) ; \(RP = 100010 \) \(\text{At this point (b) (a)(b).}
\end{itemize}

Choices: (a) the dispatch unit should stop dispatching (b) the committing should stop

Whether it is Full or Empty, the lower 5-bits of the 6-bit WP and the lower 5-bits of the 6-bit RP are identical. \(T / F\)

Say the speed of the processor went up by a factor of 2 but the speed of the memory did not change. It is noticed that the ROB is getting full more frequently now. Explain: The memory instructions (lw/sd) take more clocks of its high speed clock. Soon these instructions become bottlenecks in ROB at the committing end. In the INC (in-order commitment) design, if a lw occurs, cache miss, all junior code behind him and soon ROB becomes full. Compared to the original design, instructions are issued at double the rate but lws are not graduating any faster.
The EE557 and the EE457 MOESI state diagrams below differ in the transition from "E" state in response to the BusRd signal. Each has its advantages and disadvantages. List them.

**EE557 MOESI advantage:**

**EE557 MOESI disadvantage:**

**EE457 MOESI advantage:**

**EE457 MOESI disadvantage:**

---

**4.1 Complete the improved state diagram on the side.**

Add R/FMM as appropriate.

**Does it have advantages of both EE557 and EE457 designs above?**

None / EE557 only / EE457 only / both

**Did it avoid the disadvantages of both EE557 and EE457 designs?**

None / EE557 only / EE457 only / both

---

**4.2 In E state, and also in ______ state(s), there is only copy of the block.**

The cache tag directory is replicated (or dual ported) for both SCU and CCU to access simultaneously.

**4.3 Cache coherency protocol is implemented in (circle all applicable)**

(a) single-core single-threaded system  
(b) single-core multi-threaded system  
(c) multi-core each core running a single-thread  
(d) multi-core each core running a multiple threads

**The coherency is between (circle)**

(i) L1 and shared L2 cache  
(ii) between various (private) L1 caches

---
5 (38 points) 15 min.

Topic: Cache and Virtual memory

5.1 Branch Prediction and the BTB acting as "cache":

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


Size of the comparator: left-side: 20 bit (valid + 19 bits); right-side: 21 bit (valid + 20 bit tag)

Do you include the two zeros on the right-end of the 32-bit PC in any of the above? Yes / No

Your colleague says, "It is less likely to have two branches at different PCs (say 2040H and 6040H) with the same index (40H) which conflict 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 part of your colleague’s statement is incorrect or misleading and why?

_______________________________________________________________________________
_______________________________________________________________________________
_______________________________________________________________________________
_______________________________________________________________________________

5.2 In a 32-bit Virtual Address, 32-bit physical address, 4KB page system an 80-entry TLB with 5-way set associativity is used in the MMU. Divide the 32-bit Virtual address into Page-offset and VPN fields, and further divide the VPN into SET and TAG fields.

How many comparison units of what size (include valid bit) are needed here? ________________

5.3 The Data Memory stage in our 5-stage pipeline was divided later into three stages: TLB, cache Tag Access, Cache Tag Check. This is based on the assumption that we are using PIPT (VIPT/PIPT).

If we use the other, the number of stages would be ______ (2/3/4) namely, ___________ because ___________

5.4 MPI, standing for Miss Rate per Instruction, considers all instructions including those which do not access memory. ______ (True / False). Irrespective of the size and associativity of L1 cache vs. L2 cache, the MPI for L1 is always higher than / always lower than / has no relation to) the MPI of the L2 cache.
Nonlinear pipelines:

We need to find the Odd Parity of an 18-bit data \( D[17:0] \) using these non-linear pipelines. For each design, complete the reservation table and find ICV.

### Design #1

**Reservation Table #1**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S0</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ICV**

\[
C_4 C_3 C_2 C_1
\]

**Completion time = 7 clocks**

### Design #2

**Reservation Table #2**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S0</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ICV**

\[
C_4 C_3 C_2 C_1
\]

**Completion time = 9 clocks**

### 6.1 Greedy Initiator design

Both the *paper-pencil* design and the actual hardware design of a greedy initiator for the ICV of 1011010 (which was discussed in classnotes) are shown below.

For a different ICV of 11011001011, the paper-pencil method requires ___ OR gates, whereas the real hardware design requires ___ OR gates.

---

We enjoyed teaching this course! Hope you liked it! Hope to see some of you in EE454L or EE560. Grades will be out in a week. Enjoy your winter break!

Happy Holidays!!! - Gandhi, Jonathan, Prasanjeeet, Mehrkash, Varun, Charlene, Gourav, Zippoje, Amit, Robitkumar, Vinay, Zhenghui, Naresh, Abhilash, Naisath, Zhuioli