1 (23 points) 15 min.

Pipelining

1.1 I.F.R.F (Internally Forwarding Register File):
I.F.R.F is useful (useful/useless but harmless/harmful) in a pipelined CPU.
I.F.R.F is harmful (useful/useless but harmless/harmful) in a multicycle CPU (1st ed.).
I.F.R.F is useless but harmless (useful/useless but harmless/harmful) in a multicycle CPU (2nd ed.).
I.F.R.F is harmful (useful/useless but harmless/harmful) in a single cycle CPU.
If you marked harmful for any one, explain why it is harmful. If RF is used in pipelining
To solve the WB hazard dependency problem in pipelining, clearly it is
unnecessary in the other 3 non-pipelined designs.

5 pts

An instruction such as add $2, $2, $2 is legal. In the case of single
cycle and in the 1st ed. multicycle, since there isn’t any intervening register between
the read ports of the RF and the write port of RF, the IF RF essentially
closes the loop and causes combinatorial feedback which we call a SIN.

5 pts

1.2 If all the above four designs of the CPU (the single cycle CPU, the two multicycle CPUs, and the 5-
stage pipelined CPU) are operated at the same (clock) frequency, best performance is provided by
the Single Cycle CPU. Explain. It execute each instr. in just 1 clock. The
pipelined CPU’s goal is to achieve CPI=1 but often falls short of
the goal because of stall due to dependencies and flush due to
taken branches and jumps. The multicycle CPUs clearly take 3 to 5
clocks to finish an instruction. The reality is that the Single Cycle CPU
can not operate at high frequency.

6 pts

1.3 A Branch Delay Slot is (always advantageous / always
disadvantageous / depends on compiler’s ability to fill the slot) Explain
If the compiler puts a NOP in the delay slot, you are not gaining
anything in the case of a taken branch as you are executing an unde
NOP instead of flushing. However in the case of a failed branch, this delay
slot with a NOP only delays the execution of the fall through code.
Hence you are incurring loss either way.

5 pts

1.4 If the original 5-stage CPU of first edition did not have a delay slot for the load-word instruction, it
means, we need (need / do not need) to have a Hazard Detection Unit (HDU) to stall
an instruction in the delay slot, which is dependent on the load word.

3 pts
Pipelining (Modified Lab 7 part 3):

In the Fall 2010 midterm exam, we have an \textit{ADD4} unit in each of the two \textit{EX} stages, \textit{EX1} and \textit{EX2}. Here, those two stages, \textit{EX1} and \textit{EX2}, are merged into \textit{EX12}. So \textit{ADD8} needs an extra clock in \textit{EX12} as it has to go through the second \textit{ADD4} also.

Further we introduced a \textit{BZ} (Branch if Zero) instruction. It uses the opcode previously allocated to the \textit{SUB3} instruction. The instructions are 32-bits but the addresses are only 16-bit. \textit{PC} is 16-bit wide and is incremented by a "1". The \textit{JJJJ} in the \textit{BZ} \$S, \textit{JJJJ} stands for a 16-bit (4-digit hex) absolute branch address. If the source register \$S is a zero then branch to \textit{JJJJ} takes place \[ (PC) <= \textit{JJJJ} \text{ if } (\$S) = 0 \]. The "\textit{D}" in "\textit{4JJJJ0DS}" is a random hex digit and should not be treated as a valid destination, similar to the "\textit{DS}" in "\textit{000000DS}" for a NOP instruction. \textit{BZ} executes from the ID stage.

You need to complete the early branch mechanism (dependency stalls, branch execution by causing \textit{PC} to be changed to \textit{JJJJ} and flushing the IF stage instruction, avoiding spurious branch execution during stalling, etc.). Branch does not (and can't) execute during stalling because \textit{PC} refuses to update.

\section{2.1} Complete the design on the page next to next (on page 4).

\section{2.2} In your lab 7 Part 3 Subpart 2 (EX1 and EX2 merged case), you used the left side circuit to stall for 1 clock. Complete the design to show the \texttt{STALL} signal. Suppose you are given a flipflop with an asynchronous \textit{set} as shown in the right side below (instead of the FF with an asynchronous \textit{clear} as shown on the left). Redesign your stall circuit with this FF and show the \texttt{STALL} signal.

\section{2.3} When \texttt{STALL \_ADD8} is active, you stall the entire pipeline. True / False

When \texttt{STALL \_BR} is active, you stall the entire pipeline. True / False

\textit{IF Flush mechanism here is } \underline{different from} \underline{the wrist-band mechanism used in our pipelined CPU design}.
2.4 In this design we have implemented an early branch. Would a medium branch from EX12 be better? Yes / No It depends. Explain. Early branch reduces flush for taken branches but can potentially lose clock due to dependency so if the compiler can avoid the stall due to dependency then early branch is better. But if there are too many dependency stalls then branching thru EX12 is better because you flush (and incur penalty due to flush) only for taken branches where no penalty due to dependency stall is incurred whether branch is taken or not. Is it possible to postpone executing the BZ instruction all the way into the WB stage (WB!, not EX12)? Not possible / possible but undesirable / possible and desirable. Explain. Unlike in our MIPS pipe where we have a mem stage, we do not have any operation upstream of the WB stage which changes the state of the machine such as corrupting the register file. Hence we can postpone BZ execution all the way to WB, but then you flush 3 instructions whenever you take the branch, hence it is undesirable.

2.5 Combining EX1 and EX2 into one EX12 stage (as done here) is always worse (always better / always worse / depends on the instruction sequence in the program). Explain. If there are a series of ADD8 instructions with no interdependency, then the 5-stage CPU is better as there are no stalls. Even if there are dependencies, the 5-stage incurs as many dependency stalls as the 4-stage CPU, but not more.

2.6 How come, we carried (PC + 4) to the ID stage in the text book design, but we do not carry (PC+1) to the ID stage here? The text book design implements the MIPS relative branch requiring offset to be added to the incremented PC. Here the branches are absolute and not relative, so there is no branch target calculation.

2.7 Complete the following "Single Cycle CPU" kind of a design for the pipelined design on the next page. Complete the control unit also.
1. Complete all missing connections to
2. Complete the StALL_ADD8 logic in EX12 and StALL_BR logic in ID stage.
3. Complete all four enable (EN) controls on the pipeline registers (including PC).
4. Draw the logic to produce PCSource, IF_Flush, FORW, SKIP1, SKIP2 on this page itself.

Modified LAB 7 Part 3 Block Diagram
2.8 Now let us try to build a multi-cycle version for the design on the previous page. It is proposed that, we go for a single ALU, which can add a selected constant, \(1\) for \(\text{PC}\) and \(4\) for \(\text{ADD4}\) or \(\text{ADD8}\). \(\text{ADD8}\) uses the ALU twice to add 4 two times. This multi-cycle datapath is similar to the 1st edition design except that here the ALU is built using dynamic logic (like in the 2nd edition). There is an ALUOut register like in the 2nd edition. You need to carefully decide when to take \((\text{tap})\) data from the upstream of the ALUOut register and when to take \((\text{tap})\) data from the downstream of the ALUOut register. We have an \(\text{IR}\) register (Instruction Register) to hold the instruction at the end of the first state \((S0)\). \(\text{IR}\) is needed as \(\text{PC}\) is incremented using the ALU in the very first state. We need to support a \(\text{NOP}\) instruction here besides \(\text{MOV, ADD4, ADD8, and BZ}\).

2.8.1 Complete the datapath and the state diagram for control unit on the next two pages. To some extent, our state diagram resembles the 2nd edition state diagram reproduced below for your reference. We are doing a MOORE kind of state diagram and may be wasting a few clocks.
We do not need (need / do not need) an ALUOut_write control signal.
2.8.2 Mr. Trojan says that, we can easily improve the above state machine by combining states S1, S2, and S6 into one mealy state S126.

Complete the S126 state on the side and also write the new state transition condition from S126 to S0.
3 (43 points) 20 min.

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>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>Direct</td>
<td>A3 - A15</td>
<td>A14 - (12-bit) A3</td>
<td>A2</td>
<td>use this info!</td>
</tr>
<tr>
<td>Fully Associative</td>
<td>A3 - A3</td>
<td>A2</td>
<td>A2 - A0</td>
<td>(BE3-BE0)</td>
</tr>
<tr>
<td>Set Associative 2 Blocks/Set</td>
<td>A3 - A14</td>
<td>A13 - A3</td>
<td>A2</td>
<td>A1-A0 (BE3-BE0)</td>
</tr>
<tr>
<td>Set Associative 16 Blocks/Set</td>
<td>A3 - A11</td>
<td>A10 - A3</td>
<td>A2</td>
<td>A1-A0 (BE3-BE0)</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 \( \frac{4}{1} \) (state a number) TAG RAM(s) of size \( 4 \text{K} \times 18 \) \((17 \text{ bit tag} + 1 \text{ valid})\) together with \( \frac{1}{1} \) (state a number) comparator(s) each of \( 18 \)-bit wide.

In the case of Set Associative Mapping with 2 Blocks/Set above, we use \( \frac{2}{2} \) (state a number) TAG RAM(s) of size \( 2 \text{K} \times 19 \) \((18 \text{ bit tag} + 1 \text{ valid})\) together with \( \frac{2}{2} \) (state a number) comparator(s) each of \( 19 \)-bit wide.

In the case of Set Associative Mapping with 16 Blocks/Set above, we use \( \frac{16}{16} \) (state a number) TAG RAM(s) of size \( 256 \times 22 \) \((21 \text{ bit tag} + 1 \text{ valid})\) together with \( \frac{16}{16} \) (state a number) comparator(s) each of \( 22 \)-bit wide.

The Fully Associative Mapping is prohibitively expensive because you would need \( \frac{4096}{30} \) \((22 \text{ bit tag} + 1 \text{ valid})\) (state a number) comparator(s) each of \( 30 \)-bit wide.

In the first case of direct mapping, the main memory shall be organized in a 2-way lower-order interleaving to facilitate efficient fetching of the 2 words of a block. The main memory organization is same (same/different) in the above 4 cache organizations, because the block size is the same for all and you fetch a block irrespective of associativity in the cache.

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) a set needs to be (needs to be / does not need to be) a power of 2 in size, because the cache size is already shown as a power of 2. Such a number has only 2 as its factor and nothing else.

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):

- **Fully-associative**
- **Set-associative**
- **Direct**
4 (34 points) 20 min.

Virtual Memory:

4.1 PTBR stands for _____________________________.

6 pts

It is initiated by _______ (hardware / operating system) and is utilized by _______ (MMU / CCU) (i.e. memory management unit or cache control unit) to look up _______ (TLB / Page Table / Cache Tag RAM).

4.2 Page Table: Number of A,B,C Tables built by the OS:

PQRST on the side represents a 20-bit (5-digit hex) VPN in a 3-level page table with upper 8 bits (PQ) indexing the A-level table, next 8 bits (RS) indexing the B-level tables, and the last 4 bits (T) indexing the C-level tables.

Suppose the first 8 distinct virtual pages accessed by the application program had the VPNs as stated in TABLE-I (in sorted order).

How many tables of what size are built by OS by this time?

A-level: _____________________________________________

B-level: _____________________________________________

C-level: _____________________________________________

9 pts

4.3 The advantage of VIPT over PIPT comes from the fact that ________________

4 pts

______________________________________________________________________________

______________________________________________________________________________

4.4 Memory addresses: In a 32-bit virtual address system using 4KB pages, state any two consecutive 32-bit word addresses (in hex) which do not fall in the same virtual page. ____________

7 pts

I am evicting a page containing the byte with virtual address 23456789h. What is its virtual page number (in hex)? ____________h. What is the range of byte addresses residing in that page (lowest virtual byte address to highest virtual byte address). ____________h - ____________h

The physical page frame number in the main memory is 2 (just 2). What is the range of byte addresses residing in that page (lowest physical byte address to highest physical byte address). ____________h - ____________h

4.5 Since we use write-back only for virtual memory, we need to maintain a dirty bit associated with _______.

4 pts

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).

4.6 Fully associative mapping may not be prohibitively expensive in the case of a _______.

4 pts

(TLB / L2 cache) because ________________

______________________________________________________________________________