Overview

The following applies to all problems **unless otherwise explicitly stated.**

Consider a 2 GHz MIPS processor with a canonical 5-stage pipeline and 32 general-purpose registers. **The branch condition and jump addresses** are computed in decode. Branches are assumed not taken. The datapath implements forwarding.

The processor has 2 Gbytes of memory, 32 Kbyte direct mapped separate L1 Instruction and data caches with 64 byte lines backed by a unified 4 Mbyte 16-way set associative L2 cache with 512 byte lines and a unified 32-way 16 Mbyte L3 cache with 1024 byte lines. Applications have a 4 Gbyte virtual address space and 16 Kbyte pages. Hit time for the L1 is one cycle. On a miss in the L1, it takes 4 cycles to access the L2 cache and 1 cycle to transfer each 32-bit word from the L2 to the L1. Similarly, on a miss in the L2 cache it takes 8 cycles to access the L3 and 2 cycles for each 128-bit element (this means the L3 is accessed in chunks of 128 bits when moving data to the L2.) The memory system is banked with 16 banks each bank delivering one 32-bit word in 64 cycles. The bus to the memory system takes 2 cycles to transmit an address and is 64 bits wide, i.e., it can transfer 64 bits in a single clock cycle between the memory and the L3. The virtual memory system is supported by separate 8 entry fully associative separate TLBs for data and for instruction.

Program execution statistics are the following.

<table>
<thead>
<tr>
<th>Instruction Class</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>28%</td>
</tr>
<tr>
<td>Store</td>
<td>15%</td>
</tr>
<tr>
<td>ALU</td>
<td>40%</td>
</tr>
<tr>
<td>Branches</td>
<td>12%</td>
</tr>
<tr>
<td>Jumps</td>
<td>5%</td>
</tr>
</tbody>
</table>
1. **(15 pts)** Consider execution of a MIPS code sequence on a pipelined data path shown on the next page.

   a. Show the modifications required to the pipelined datapath overleaf to implement support for the \texttt{jal} instruction. \textbf{To receive credit you must be clear.}

   There is more than one solution. The solution below has three functional additions.

   1. Compute the jump address: Take the least significant 26 bits of the instruction, multiply by 4 (convert word address to byte address), concatenate the most significant 4 bits of the PC to generate a 32 bit address. This address is now a new input to the \texttt{PCSrc} mux. The controller generates the proper PCSrc mux control bits.

   2. Add a third input to the \texttt{RegDst} mux hardwired to 31. Mux control is now 2 bits.

   3. Make PC+4 available through \texttt{WB} and as a new input to the \texttt{MemToReg} mux. Expand the number of bits that controls this mux.

   4. Controller is updated to include a flush of IF/ID and generate a \texttt{jal} signal.

   5. Since register 31 is not written until \texttt{jal} reaches WB, the procedure must have at least one instruction before executing a \texttt{jr} instruction.

   b. What would be the values of all of the control signals to realize this implementation.

   Note that some of these control signal values can be don’t care

   \begin{center}
   \begin{tabular}{|c|c|c|c|c|c|c|c|c|}
   \hline
   Instr. & RegDst & AluSrc & MemToReg & RegWrite & MemRead & MemWrite & Branch & AluOp \\
   \hline
   jal    & 10     & 0      & 10       & 1        & 0       & 0        & 0       & 00       \\
   \hline
   \end{tabular}
   \end{center}
c. Now consider the execution of the following code sequence – specifically the execution of the function `func`: Why will execution be incorrect in the datapath described on page 2 and what can you do to ensure correct execution? Assume support for the `jr` instruction is similar to that of the `jal` instruction and `addi` and `move` are supported similar to the `add` instruction.

```
.data
start: .word 0x32, 33, 0x34, 35

.text
la $a0, start
addi $a1, $a0, 0x10
jal func
li $v0, 10
syscall
func: add $t7, $zero, $zero
loop: lw $t6, 0($a0)
     add $t7, $t7, $t6
     addi $a0, $a0, 4
     bne $a1, $a0, loop
     move $v0, $t7
     jr $31
```

Since branches are resolved in decode `bne` will not receive the correct forwarded value. Therefore we must either insert 2 stall cycles or 2 nops between `addi $a0, $a0, 4` and `bne $a1, $a0, loop`.

There is also a load-to-use hazard on the `lw` instruction that must either be supported in hardware or utilize a `nop`. 
2. (10 pts) We wish to implement an atomic swap instruction – swap r1, offset(r2). This instruction atomically swaps the contents of a register with a location in memory.

a. What does it mean for the execution of this instruction to be atomic?

The execution of the instruction is atomic, if all of its actions are completed in their entirety without any intervening actions (read or write) to the state that is being accessed. For example, the increment of a memory location is atomic if when once begun, there is no intervening access to the memory location until the increment operation has completed.

b. Using a few instructions, provide a MIPS assembly language implementation of this instruction using the following MIPS instructions.

load linked: ll r1, offset(r2)
store conditional: sc r1, offset(r2)

The following code sequence atomically swaps the contents of register $s4 and the memory location whose address is in register $s1.

```
loop: add $t0,$zero,$s4
       ll $t1,0($s1) ;load linked
       sc $t0,0($s1) ;store conditional
       beq $t0,$zero,loop ;branch store fails
       add $s4,$zero,$t1 ;put load value in $s4
```
3. (30 pts) Consider the memory system.

   a. (5 pts) Show the breakdown of the following addresses: i) virtual address, ii) physical address, iii) physical address as interpreted by the L1 cache, iv) physical address as interpreted by the L2 cache, and v) physical address interpreted by the L3 cache. **Be clear!**

   The most significant bit of the Physical Address will be zero if the address space is only 2 Gbytes. This would apply to the address breakdowns below.

<table>
<thead>
<tr>
<th>PA</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td></td>
<td>14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VA</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td></td>
<td>14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L1</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>9</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L2</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>9</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L3</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>
b. (5 pts) Assuming a data miss in the L1 and L2 but a hit in the L3 what is the miss penalty in cycles?

L2 hit service to L1 = 4 + 16 * 1 = 20 cycles  
L3 hit service to L2 = 8 + (128/4) * 2 = 72 cycles  

Note: A L2 cache line is 512 bytes = 128 words. It takes 2 cycles to transmit 4 words (128 bits) from the L3 to the L2. There is an initial cost of 8 cycles.

Therefore the miss penalty is 20 + 72 cycles = 92 cycles. Note that this is not the average hit time. This is only the miss penalty when there is a miss in the L1 and L2 and data is served from the L3.

c. (5 pts) Assuming a Dirty bit and a Valid bit for each tag entry, what is the size in bits of each of the following

Note that each tag is 17 bits since we are assuming full 32-bit addresses

L2 Directories: (2^9 * 16) * 2^4 = 2^{17} bits  
Note that there are 16 ways and each entry has 14 + 1 + 1 = 16 bits.
d. **(8 pts)** The instruction and data miss rates in the L1 are 2% and 4% respectively. The unified local miss rate in the L2 is 25% and the L3 is 44%. What is the worst-case increase in CPI due to memory accesses?

Main memory miss penalty = \((2 + 64 + \frac{16}{2} \times 1) \times 16 = 1184\) cycles

Note that transmission of an address takes 2 cycles while two words can be transmitted from the memory system to the cache in 1 cycle. Each memory cycle delivers 16 words. 16 memory cycles are necessary to deliver a cache line. From the previous sub-problems we already have the miss penalties between the other layers of the cache.

\[
\begin{align*}
&\text{CPI increase due to instruction misses} = 0.02 \times (20 + 72 \times 0.25 + 0.25 \times 0.44 \times 1184) \\
&\text{CPI increase due to data misses} = 0.04 \times 0.43 \times (20 + 0.25 \times 72 + 0.25 \times 0.44 \times 1184)
\end{align*}
\]

e. **(7 pts)** Provide a sequence of data references (addresses) to the L1 data cache that will generate a miss on every reference.

Addresses 0x00048c0 and 0x800048c0 will map to the same line in the L1 data cache. An alternating sequence of references to these addresses will create a miss on every reference.
4. (10 pts) Base CPI of the core is 1.0 and 40% of loads produce a load-to-use hazard and 80% of these can be filled with useful instructions while 30% of branches are taken. We have separate direct mapped I and D caches (IF and MEM stages). If we change the L1 cache designs from direct mapped to two-way set associative, we will reduce the L1 instruction miss rate by from 2% to 1.6% and the L1 data miss rate by from 4% to 3%. However, we will need to add a pipeline stage to each of IF and MEM. Assuming every L1 miss hits in the L2, is the change to a set associative L1 worth it? Justify your answer.

\[
\text{CPI-increase} = 0.3 \times 0.12 \times 1 + 0.4 \times 0.25 \times 0.2 \times 1 + 0.02 \times 20 + 0.04 \times 0.43 \times 20 \\
\quad \text{(branches + ALU hazards + instr miss + data miss)} \\
\text{CPI-increase-opt} = 0.3 \times 0.12 \times 2 + 0.4 \times 0.25 \times 0.2 \times 2 + 0.016 \times 20 + 0.03 \times 0.43 \times 20
\]

\[
E_1 = #I \times \text{CPI-increase} \times \text{clock}\_cycle \\
E_2 = #I \times \text{CPI-increase-opt} \times \text{clock}\_cycle
\]

Compare \(E_1\) and \(E_2\) to determine if this is a good idea.
5. **(10 pts)** Consider a 32x32 matrix stored in memory in row major form, i.e., all elements in the first row followed by all elements in the second row, and so on. Each element is an integer number that takes 4 bytes. The matrix is stored starting 0x10010000.

   a. **(5 pts)** In which L2 set can we find the element A(2,4)? The first element is A(0,0) and is stored at memory location 0x10010000.

      The byte address of the first element of row 2 is 0x10010000 – the first two rows (0 and 1) have 64 elements or 256 bytes. 256 = 0x100. The 4th element of row 2 is 4*4 = 0x10 bytes offset from the first element. Therefore the address of A(2,4) is 0x10010110.

      The L2 has a line size of 512 bytes (9 bit offset). The number of sets in the L2 = \( \frac{2^{22}}{2^9 * 2^4} \) = 512 (9 bits). The 9 bit set index value in address 0x10010110 is 0x080.

   b. Consider two threads working on two different halves of the matrix. Each thread executes on a different core. How would assign the matrix elements between the two threads so the one thread never accesses a cache line that contains elements belonging to the other thread, i.e., a cache line can only have elements that are processed by one thread.

      Consider the L1 cache since all other cache line sizes are larger and multiples of the L1 cache line size, and accesses by the core are to the L1 cache. Each line is 64 bytes or 16 words. Therefore, each row of the matrix (32 elements) is stored in two successive cache lines. The first 16 rows of the matrix are stored in 32 (16 *2) consecutive cache lines, while the remaining 16 rows are stored in the next 32 cache lines. Thread 1 should be allocated to the first 16 rows. Thread 2 should be allocated to the next 16 rows. Now each thread will never access cache lines accessed by the other thread.
6. (15 pts) Consider a bus-based cache-coherent multiprocessor with four cores. Each core has a 1Kbyte fully associative cache with 256 byte lines. Main memory is 64Kbytes. The following figures show the state of each cache line in each cache and the corresponding tag. An invalid line (INV) is simply an empty line and a miss always fills an invalid line before evicting a valid line (V). A valid line means multiple copies may in the cache and are not dirty. If a cache has the only copy (which can also be dirty) the state of the line is EX.
   a. To keep the caches coherent, what could be the state of the caches after i) Processor 0 generates a read to address 0x1300, ii) processor 3 generates a write to address 0xCD40, and iii) core 1 generates a write to address 0xC104. You need only show changes.

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
<th>Core 2</th>
<th>Core 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>V 13</td>
<td>EX C1</td>
<td>V 13</td>
<td>V 9B</td>
</tr>
<tr>
<td>INV CD</td>
<td>INV CD</td>
<td>EX FE</td>
<td>V 00</td>
</tr>
<tr>
<td>V 12</td>
<td>INV AA</td>
<td>V A3</td>
<td>V 12</td>
</tr>
<tr>
<td>V 33</td>
<td>V D0</td>
<td>INV 22</td>
<td>EX CD</td>
</tr>
</tbody>
</table>

b. Can a write through cache design be used to solve the coherence problem? How or why not?

No. Write through does not affect copies of data in another cache. Those copies will remain stale (and incorrect).
7. **(10 pts)** This question relates to the disk subsystem.

a. The disks have a controller with 0.5ms overhead and a transfer rate of 100 Mbytes/sec at 7200rpm with average seek times of 6 ms. We have to make a choice between 4Kbyte and 256Kbyte pages. What is the difference in page fault service times between these page sizes?

   The only elements of the disk access latency that varies as a function of the page size are the transfer rate.

   \[
   \text{Latency} = \text{controller\_overhead} + \frac{1}{2} \text{rotational delay} + \text{average\_seek\_time} + \text{transfer\_time}
   \]

   \[
   \text{transfer\_time} = \text{page\_size} \times 10^{-8} \text{ (secs/byte)}
   \]

b. Consider power gating (turning off) the parity disk in a RAID 4 organization. Under what conditions would you have to power up the parity disk?

   Whenever a write operation occurs, the parity disk will have to be updated and therefore powered up. Under continuous read operations, it can remained powered off. It also is powered up on a failure when the contents of the failed disk has to be reconstructed.