Midterm Examination
CDA 4101 Computer Organization
Spring 2000
March 1, 2000
Duration: 75 minutes

<table>
<thead>
<tr>
<th>No.</th>
<th>Topic</th>
<th>Possible</th>
<th>Awarded</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Basic Facts</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Basic Facts</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Control/data path</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Hazard stalls</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Branches</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

Total Awarded:

NAME: SOLUTIONS
Problem 1: 20 points

The answers to each of the following should be brief and to the point (no more than a sentence or two).

(a) 3 pts. What is the difference between combinational logic and sequential logic?
   Solution: Combinational: current output depends only on current input. Sequential: current output depends on current and previous output via internal state, i.e., it has memory.

(b) 3 pts. What determines the cycle time, \( \tau_1 \), of a single-cycle implementation of a MIPS processor?
   Solution: The path with the longest delay through the combinational logic that updates the machines state stored in memory and registers.

(c) 3 pts. What determines the cycle time, \( \tau_2 \), of a multiple-cycle implementation of a MIPS processor?
   Solution: The path with the longest delay through the combinational logic that is active in any one state of the multicycle implementation.

(d) 3 pts. Which would you expect to be smaller and why, \( \tau_1 \) or \( \tau_2 \)?
   Solution: \( \tau_2 \) is usually smaller since the combinational logic that is active in any one state is usually a single piece of the combinational logic that makes up a single cycle implementation.

(e) 4 pts. Describe an example of a single program running on two different implementations of the MIPS architecture where the implementation that takes longer to complete the program has a larger MIPS performance metric for that program.
   Solution: The homework problem in textbook exercise 2.15.

(f) 4 pts. Describe an example of two programs to solve the same problem running on the same implementation of the MIPS architecture where the program that takes less time has a lower MIPS performance metric rating.
   Solution: This is typically the case any time that there is an algorithm that can reduce the operation count by exploiting the problem’s structure. For example, if the number of floating point operations can be reduced in an algorithm by exploiting the fact that many of the data elements are actually 0 and do not contribute to the solution. Extra complexity is needed in the data structures and their accesses since only nonzeros are stored but the number of floating point operations on such data usually drops significantly reducing time but also reducing the performance metric of MFLOPS (million floating point operations per second.)
Problem 2: 25 points

(a) 5 pts. Assuming peak performance of the basic MIPS pipeline used in the text, how many cycles would a code that executes 25,000 instructions take to complete?

Solution: Peak performance implies no stalls encountered. The standard pipeline implementation discussed in the text has 5 stages. Therefore the first operation is done after 5 cycles and another completes on each cycle thereafter. Time is therefore,

\[ t = 5 + (25000 - 1) = 25004 \]

(b) 10 pts. Suppose the workload on your machine spends 90% of the time in floating point arithmetic. A vendor proposes that you upgrade your machine to perform the floating point arithmetic 1000 times faster. Since you are such a good customer the vendor suggests a price of only 100 times the price of your current system. Do you buy the upgrade? Justify your answer.

Solution: Amdahl’s law could be used on this problem but there is an easier solution. The new time is, given a speedup of \( R \) in the floating point, \( t = (0.1 + 0.9/R)t_{old} \). Note that even if \( R = \infty \) \( t \geq 0.1t_{old} \).

Therefore we can bound speedup,

\[ S = \frac{t_{old}}{t} \leq \frac{t_{old}}{0.1t_{old}} \leq 10 \]

So a price of 100 times the current system for a benefit that is no more than a factor of 10 is unacceptable and the upgrade should not be purchased.

(b) 10 pts. The `bned` instruction is a branch on not equal with a single delay slot. Rearrange the code below to minimize completion time on the basic MIPS pipeline.

```
loop:   lw  $10,0($2)
        mul $11,$10,$5
        sw $11,0($2)
        addi $2,$2,4
        bned $2,$4, loop
```
**Solution:** There is a stall here that we need to worry about and we must fill the branch delay slot.

The relative ordering of the lw, mul and sw cannot be altered due to a dependence string. However the addi can be moved into the load-use stall slot. Since this alters the order of the addi and the sw that uses $2$ for its address computation we must alter the offset in the sw to $-4$. The sw can also be used to fill the delay slot.

```
loop:   lw   $10,0($2)
       addi $2,$2,4
       mul $11,$10,$5
       bned $2,$4, loop
       sw $11,-4($2)
```
Problem 3: 15 points

Consider the datapath and control lines for the single-cycle implementation including the jump instruction in the figure on the next page (Textbook figure 5.29). Draw the additions to the datapath and control on this figure needed to implement the jal instruction. Also fill in the appropriate settings of the control lines in the table and modify or extend the existing ones as needed.

Recall, the jal instruction is the jump and link used to implement a function call. It has the same format as the jump (j) instruction. It transfers control to the specified address, like a jump, but also places PC + 4 in $31.

Solution: There are two basic additions to the datapath needed. Since we are writing $PC + 4$ into $31$. We must be able to get 31 to the write register lines of the register file. These are the output lines of the MUX controlled by the single bit RegDst control line. We must also get $PC + 4$ to the write data lines of the register file. These are the output lines of the MUX controlled by the single bit MemtoReg control line.

The first is easily handled by adding lines with 31 as constant binary input (like the 4 is constant input to the add unit) to the MUX controlled by RegDst. The MUX therefore is expanded to a three-input MUX. This implies that RegDst must be expanded to a two-bit control signal from its current one bit.

The second is accomplished by adding a line from anywhere along the $PC + 4$ line that already exists to an input line of the MUX controlled by MemtoReg. The MUX therefore is expanded to a three-input MUX. This implies that MemtoReg must be expanded to a two-bit control signal from its current one bit.

These new control bits must have their assignment specified for the jal instruction as well as all the others that use them. The new control signals are given in the table below with the changes in bold face. We have chosen to use the pattern 10 to select the new input line from the new three-input MUX’s.

<table>
<thead>
<tr>
<th>Inst.</th>
<th>Reg Dst</th>
<th>ALU Src</th>
<th>Memto Reg</th>
<th>Reg Write</th>
<th>Mem Read</th>
<th>Mem Write</th>
<th>Branch</th>
<th>Jump</th>
<th>ALU Op1</th>
<th>ALU Op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>01</td>
<td>00</td>
<td>00</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lw</td>
<td>00</td>
<td>1</td>
<td>01</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sw</td>
<td>XX</td>
<td>1</td>
<td>XX</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>beq</td>
<td>XX</td>
<td>0</td>
<td>XX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>jal</td>
<td>10</td>
<td>X</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
5.29 here
Problem 4: 20 points

Consider the 5 stage implementation of the MIPS pipeline where the target address and branch condition resolution are performed during the ID stage. Assume that forwarding logic has been added to forward to branch instructions such as beq in the ID stage.

Recall, that in some cases hazards force the generation of stalls to place the instructions in the appropriate stages of the pipeline simultaneously so forwarding or the register file can handle the data dependence. The load-use scenario was presented, i.e., a load followed immediately by a consuming instruction, as was the scenario of an R-type instruction followed by a branch where the R-type’s destination is one of the source registers for the branch, e.g., add $1,$2,$3 followed by beq $1, $4, loop.

(a) 10 pts. Consider the interaction of a load and a beq on this implementation of the pipeline. Describe the conditions under which one or more stalls must be generated. Illustrate them with a timing diagram.

Solution:

There are two situations that cause stalls when a lw and beq interact. If the lw and beq are separated by 0 or 1 (independent) instruction 2 cycle and 1 cycle stalls are required respectively. This is easily seen by the following timing diagrams with the stalls added. Recall, forwarding is assumed so the MEM stage must have completed for the lw before the beq can receive the data in its ID stage.

\[
\begin{align*}
\text{lw} & \quad F \quad D \quad X \quad M \quad W \\
\text{beq} & \quad F \quad D \quad D \quad D \quad X \quad M \quad W \quad 2 \text{ stalls required}
\end{align*}
\]

\[
\begin{align*}
\text{lw} & \quad F \quad D \quad X \quad M \quad W \\
\text{any inst.} & \quad F \quad D \quad X \quad M \quad W \\
\text{beq} & \quad F \quad D \quad D \quad X \quad M \quad W \quad 1 \text{ stall required}
\end{align*}
\]
(b) 10 pts. Derive the logical conditions for the stall hardware to evaluate in order to correctly handle the problem. Be specific about timing and the location of the signals used.

Solution:
Both stall situations can be handled by having the instruction currently in the ID stage check conditions relative to the instructions currently in the EX and MEM stages. In this way, on any cycle only one stall cycle is generated, i.e., we do not ever conclude that two stalls are needed. This would complicate the implementation of the stall generator needlessly.

The conditions checked are:

- the instruction in the ID stage is a beq

- AND

  - the instruction in the MEM stage is a lw with a destination register that is the same as one of the sources of the instruction in the ID stage

  - OR

  - the instruction in the EX stage is a lw with a destination register that is the same as one of the sources of the instruction in the ID stage

If this evaluates true a stall is generated. Note that the situation of a lw followed immediately by a beq does in fact generate two stall cycles since this test will succeed on two successive cycles each of which generates a single stall.

The information on the instructions in the ID, EX and MEM stages is found in the IF/ID, ID/EX, and EX/MEM registers respectively. The only exception to this is the fact that the ID stage has a beq. The instruction has not been decoded so it is necessary to examine the op code in the IF/ID register using combinational logic to detect a beq in the ID stage.
Problem 5: 20 points

Consider an implementation of the MIPS pipeline that:

1. has 10 stages,
2. uses delay slots,
3. uses, as well, simple branch prediction (1 bit is generated to predict taken or not taken, a branch target buffer is not used, i.e., no target address generated in the prediction),
4. uses cancellation to avoid erroneously updating the state on a misprediction,
5. computes the target address of a beq during the fifth stage of the pipeline
6. and resolves the branch condition during the eighth stage of the pipeline.

(a) How many delay slots are there? Solution: 4

(b) If the beq is fetched during clock cycle 1 (CC1), during what clock cycle do the nops appear in the pipeline as a result of cancellation due to misprediction? Solution: CC 9

(c) What is the maximum number of instructions that are cancelled by any single misprediction? Solution: 3

(d) Assuming that cancellation (flushing) is the only mechanism to avoid the erroneous update of the program’s state by a mispredicted branch of control, what stages of the pipeline absolutely cannot update the registers or the memory? Solution: stages 1, 2, and 3
Justification: The timing diagram is filled in to demonstrate a misprediction. Consider a prediction of taken on the beq. Since the prediction does not give a target address the predicted control path cannot be fetched until CC 6 (the target is produced in CC 5). The cycles CC 2, CC 3, CC 4, and CC 5 can fetch instructions or be stalls. Since the prediction was “taken” there is no point in fetching the instructions following the beq in the code if they correspond to “not taken”. Therefore, the instructions fetched in CC 2, CC 3, CC 4, and CC 5 should be 4 delay slot instructions that follow the beq in the code and are always executed, i.e., a delay slot is never cancelled based on actual branch behavior versus predicted branch behavior.

The instructions on the predicted control path that are fetched in CC 6, CC 7 and CC 8 are labelled P1, P2, and P3 to denote predictions. If the prediction was incorrect then these need to be cancelled on the cycle following the cycle in which the branch resolution was computed. This means that P1, P2, and P3 will all be cancelled on CC 9 as illustrated. Actually, 3 is the maximum number of instructions that can be fetched in this interval before cancellation is possible. It could be fewer if there is a dependence between any of the P1 that would cause a stall. The instructions in the “true” control path can also be fetched at that point – T1 and T2 in the diagram.

Note that during CC 6, CC 7 and CC 8 instructions P1, P2 and P3 had exercise stages 1, 2, and 3 of the pipeline. Since they cannot be cancelled in time to stop any state update in those stages we know that stages 1, 2, and 3 cannot be allowed to update memory or registers in the implementation of the pipeline described here.

Note the description above started with the assumption that the prediction was “taken” in order to justify the delay slots that are assumed to be used. If the prediction was “not taken” it is certainly true that the predicted path could be fetched starting in CC 2. If the prediction was incorrect the predicted instructions (7 of them) could be cancelled in CC 9 as described above. This would argue for no delay slots. Recall, however, that the definition of a delay slot cannot be made at runtime since it corresponds to an instruction set architecture constraint on the way in which the code is interpreted. The instructions that follow the beq in the text of the code must be known to be delay slots (not cancellable) or not delay slots (cancellable) at the time the code is generated. Their cancellability cannot be variable. Therefore, in this pipeline even a prediction of “not taken” does not begin to fetch the predicted path of control until after the target address is available and the 4 instructions following the beq in the text of the code must be interpreted as delay slots.
$S_i$ in a cell indicates that the instruction in that column is active in the $i$-th stage of the pipeline during the clock cycle corresponding to the row of the table.

<table>
<thead>
<tr>
<th>Cycle</th>
<th>$beq$</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC1</td>
<td>S1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC2</td>
<td>S2</td>
<td>S1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC3</td>
<td>S3</td>
<td>S2</td>
<td>S1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC4</td>
<td>S4</td>
<td>S3</td>
<td>S2</td>
<td>S1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC5</td>
<td>S5</td>
<td>S4</td>
<td>S3</td>
<td>S2</td>
<td>S1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC6</td>
<td>S6</td>
<td>S5</td>
<td>S4</td>
<td>S3</td>
<td>S2</td>
<td>S1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC7</td>
<td>S7</td>
<td>S6</td>
<td>S5</td>
<td>S4</td>
<td>S3</td>
<td>S2</td>
<td>S1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC8</td>
<td>S8</td>
<td>S7</td>
<td>S6</td>
<td>S5</td>
<td>S4</td>
<td>S3</td>
<td>S2</td>
<td>S1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC9</td>
<td>S9</td>
<td>S8</td>
<td>S7</td>
<td>S6</td>
<td>S5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>S1</td>
<td></td>
</tr>
<tr>
<td>CC10</td>
<td>S10</td>
<td>S9</td>
<td>S8</td>
<td>S7</td>
<td>S6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>S2</td>
<td>S1</td>
</tr>
<tr>
<td>CC11</td>
<td>S10</td>
<td>S9</td>
<td>S8</td>
<td>S7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>S3</td>
<td>S2</td>
<td></td>
</tr>
<tr>
<td>CC12</td>
<td>S10</td>
<td>S9</td>
<td>S8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>S4</td>
<td>S3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC13</td>
<td>S10</td>
<td>S9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>S5</td>
<td>S4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC14</td>
<td>S10</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>S6</td>
<td>S5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC15</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>S7</td>
<td>S6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC16</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>S8</td>
<td>S7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC17</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>S9</td>
<td>S8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC18</td>
<td>-</td>
<td>S10</td>
<td>S9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC19</td>
<td>-</td>
<td>-</td>
<td></td>
<td>S10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>