Solutions for Homework Chapter 5
Computer Organization Spring 2002

Due date: 2:30 PM, Monday, 2/25/02, TA responsible: Y. Abbas

Problem 1 (30 points)

Consider the single-cycle implementation given in Figure 5.29. You are to add an addi instruction, i.e., an
add with immediate data. Indicate any changes to the datapath that must be done, e.g., extra wires, extra
logic, extra registers, etc. Also specify the values of all of the control lines that control the datapath and
guarantee correct execution. Be sure to justify your answer.

Solution:

For the addi instruction we need a path that takes the rs field from the instruction uses it as a register
number in the register file to read data and send that data to the upper input of the ALU. Such a path
already exists. In fact, no control lines need to be set to make that path since it exists in the datapath as
defined in Figure 5.29 for all instructions. (It may not be used for all instructions of course.)

We also need a data path from the immediate operand field in the instruction to a sign extender and
then to the lower input on the ALU. This path also exists in the datapath of Figure 5.29 but we must set
the control lines in order to make sure it is followed. We must also set the control lines to tell the ALU to
perform an addition. Since addi is an I type this must be done by setting the two bit ALUOp control.
The control signals required so far are,

\[
\begin{align*}
ALUSrc & = 1 \ldots \text{to choose immediate operand} \\
ALUOp & = 00 \ldots \text{to add the immediate value to the rs register.}
\end{align*}
\]

The datapath in Figure 5.29 sends the output of the ALU (that we now have set to produce the result of
the addi) to the MUX controlled by MemToReg. To set the feedback path to the register file write-data lines
to the ALU output and to write the destination register specified in the rt field, we must also set RegWrite
and RegDst appropriately. These settings are

\[
\begin{align*}
MemToReg & = 0 \ldots \text{to set ALU output as feedback to write-data} \\
RegWrite & = 1 \ldots \text{to write register file} \\
RegDst & = 0 \ldots \text{to select the rt field as the destination register.}
\end{align*}
\]

Finally, we must avoid contaminating the state by indicating to the memory that no transaction is to be
performed and setting the PC to be updated to PC + 4. This is accomplished by the settings

\[
\begin{align*}
Branch & = 0 \\
Jump & = 0 \\
MemRead & = 0 \\
MemWrite & = 0
\end{align*}
\]

Problem 2 (30 points)

Textbook problem 5.14. Use the single cycle implementation in Figure 5.29 for this problem. Be sure to
provide details of the derivation of your answer, i.e., be specific about why you have chosen a particular
critical path through the machine.

Solution:

To solve this problem we have to consider all the parallel paths that are followed during the execution
of an instruction. The path that has the longest settling time determines the cycle time. It is convenient to
organize the consideration of the combinational paths by considering each instruction in turn.

First we consider the flow of information for the the load and determine the settling time for the paths
as a function of X and Y.
Clearly there are three paths of interest by which the state of the code can be updated. The PC is updated via the PC adder and operands from the PC and a constant 4.

\[ PC \rightarrow PC + 4 \text{ adder } \rightarrow PC \]

The time for this path is \( X \text{ ns} \)

The data is loaded by a path that fetches the instruction, computes the address using the ALU, accesses data memory, and updates the register file:

\[ PC \rightarrow IM \rightarrow \text{RegFile} \rightarrow ALU \rightarrow Data\text{Mem} \rightarrow \text{RegFile} \]

The time for this path is \( 2 + 1 + 2 + 1 = 8 \text{ns} \)

The store instruction also uses the PC update path and a path that is included in the load path but does not require the final update of the register file to yield a store data path time of \( 7 \text{ns} \).

The R type instructions also uses the PC update path and a path that is included in the load path but does not require the access to data memory to yield a R type path time of \( 6 \text{ns} \).

The \( \text{beq} \) instruction uses the BEQ adder to compute a target address. This requires two operands via two paths that meet at the BEQ adder. The first path is that of the immediate operand giving the offset.

\[ PC \rightarrow IM \rightarrow \text{BEQ adder} \]

The second is that of the \( PC + 4 \) operand

\[ PC \rightarrow PC \text{ adder } \rightarrow \text{BEQ adder} \]

The result then returns to the PC. The time for this is \( \max(Y + 2, X + Y) \) reflecting the time for the first and second paths respectively and the settling of the BEQ adder.

The \( \text{beq} \) also has a settling time for the evaluation of the condition. This path is

\[ PC \rightarrow IM \rightarrow \text{RegFile} \rightarrow ALU \rightarrow MUX \]

It has a settling time of \( 5\text{ns} \).

So as a function of \( X \) and \( Y \) we have the following times to consider

1. \( X \text{ ns} \)
2. \( 8\text{ns} \)
3. \( 7\text{ns} \)
4. \( 6\text{ns} \)
5. \( Y + 2\text{ns} \)
6. \( X + Y \text{ns} \)
7. \( 5\text{ns} \)

Of course all paths are active for all instructions. Depending on the instruction the path may or may not produce information that is used to update the state. However, all paths that update the state must produce useful information for at least one instruction, otherwise, it would not be in the data path. As indicated in the text discussion we have ignored transmission time along the connections indicated in the figure and MUX settling times, and control signal generation time. As a result this is a very simple timing model. But it does give some indication of how the cycle time is determined in a single cycle machine.

Now we can consider each of the cost configurations listed in the problem.

- When \( X = Y = 3\text{ns} \), the longest path and therefore the cycle time is \( 8\text{ns} \)
- When \( X = Y = 5\text{ns} \), the longest path and cycle time is \( X + Y = 10 \text{ ns} \).
- When \( X = 1\text{ns} \) and \( Y = 8 \text{ ns} \), the longest path and cycle time is \( Y + 2 = 10 \text{ ns} \).
Problem 3 (40 points)

Textbook problem 5.24. However, instead of using the data from Figure 4.54 use the data below.
You have a benchmark code such that the instructions executed have the following statistics:

<table>
<thead>
<tr>
<th>instruction</th>
<th>percentage of total instructions executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>load</td>
<td>21%</td>
</tr>
<tr>
<td>store</td>
<td>12%</td>
</tr>
<tr>
<td>add</td>
<td>36%</td>
</tr>
<tr>
<td>sub</td>
<td>10%</td>
</tr>
<tr>
<td>j</td>
<td>10%</td>
</tr>
<tr>
<td>beq</td>
<td>11%</td>
</tr>
</tbody>
</table>

Using the given data we can compute the percentages for the various instruction types as follows:

\[
\text{Load} = 21\% \\
\text{Store} = 12\% \\
\text{R-type} = 46\% \\
\text{J/Branch} = 21\%
\]

Let the instruction count \(= 100\) for simplicity. (This is essentially the same as doing everything relative to an arbitrary total operation count.) To determine which machine is fastest we must determine either the execution time of the benchmark or the performance for each machine.

We have from our earlier work

\[
T = CPI \times IC \times CT
\]

where \(T\) is execution time, \(CPI\) is the number of cycles per instruction, and \(CT\) is the cycle time. So if \(CT\) is in seconds per cycle then \(T\) is in seconds.

The value of \(CPI\) must take into account the mix of instructions, i.e., it is the average number of cycles per instruction for the benchmark. This is machine dependent since each of the machines have different numbers of cycles for the different types of instructions. By considering the description of each machine and which states have been combined we can deduce the effect on the number of cycles per instruction type. These cycle counts can then be weighted by the percentage of the total number of instructions each type constitutes to get an average \(CPI\) for the benchmark on each machine. These results are summarized below.

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>%</th>
<th>M 1 cycles</th>
<th>M 2 cycles</th>
<th>M 3 cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loads</td>
<td>21%</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Stores</td>
<td>12%</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>R-type</td>
<td>46%</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>J/Branch</td>
<td>21%</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>average CPI</td>
<td>100%</td>
<td>4.0</td>
<td>3.33</td>
<td>3</td>
</tr>
</tbody>
</table>

To get the cycle time \(CT\) for each machine we must convert the clock rate supplied for each machine.

We have

- \(M1\) has a clock rate of 500 MHz and therefore \(CT = 2\)ns.
- \(M2\) has a clock rate of 400 MHz and therefore \(CT = 2.5\)ns.
- \(M3\) has a clock rate of 250 MHz and therefore \(CT = 4\)ns.

We have the following execution times

- \(M1 - T = 4 \times 2 \times 100 = 800\)ns
- \(M2 - T = 3.333 \times 2.5 \times 100 = 833\)ns
- \(M3 - T = 3 \times 4 \times 100 = 1200\)ns

Or in terms of performance measured in MIPS (millions of instructions per second) we have

3
• $M1$ – 125 MIPS
• $M2$ – 120 MIPS
• $M3$ – 83 MIPS

As we can see from the calculations above, M1 is the fastest for the given instruction mix of the benchmark code.

There are of course many answers to the second part of the question. One observation that is seen from the data in the table is that for M3 the CPI does not depend on the type of instruction. M3 also reduces the CPI for loads from 5 (for M1) to 3. From this we can easily conclude that for a instruction mix containing a high percentage of loads, M3 will be faster than M1.