Theory Assignment # 2

Instructions

- Due Date for Assignment is **01-01-2013**.
- No assignment will be accepted after the due date.
- You must complete your work **individually** and you are not allowed to **copy** your fellow student's work

Task

You are to solve the following end problems of Chapter 6 (Problems attached with this file).

Students with **Odd** Roll no should solve Question no

6.1	6.3	6.14	6.23

Students with **Even** Roll no should solve Question no

6.2	6.4	6.17	6.18

With remarkable advances in processing, Amdahl's law suggests that another part of the system will become the bottleneck. That bottleneck is the topic of the next chapter: the memory system.

An alternative to pushing uniprocessors to automatically exploit parallelism at the instruction level is trying multiprocessors, which exploit parallelism at much coarser levels. Parallel processing is the topic of Chapter 9, which appears on the CD.

6.13 Historical Perspective and Further Reading

This section, which appears on the CD, discusses the history of the first pipelined processors, the earliest superscalars, the development of out-of-order and speculative techniques, as well as important developments in the accompanying compiler technology.

6.14 Exercises

6.1 [5] <§6.1> If the time for an ALU operation can be shortened by 25% (compared to the description in Figure 6.2 on page 373);

a. Will it affect the speedup obtained from pipelining? If yes, by how much? Otherwise, why?

b. What if the ALU operation now takes 25% more time?

6.2 [10] <§6.1> A computer architect needs to design the pipeline of a new microprocessor. She has an example workload program core with 10^6 instructions. Each instruction takes 100 ps to finish.

a. How long does it take to execute this program core on a nonpipelined processor?

b. The current state-of-the-art microprocessor has about 20 pipeline stages. Assume it is perfectly pipelined. How much speedup will it achieve compared to the nonpipelined processor?

c. Real pipelining isn't perfect, since implementing pipelining introduces some overhead per pipeline stage. Will this overhead affect instruction latency, instruction throughput, or both?

6.3 [5] <§6.1> Using a drawing similar to Figure 6.5 on page 377, show the forwarding paths needed to execute the following four instructions:

```
add $3, $4, $6
sub $5, $3, $2
lw  $7, 100($5)
add $8, $7, $2
```

6.4 [10] <§6.1> Identify all of the data dependencies in the following code. Which dependencies are data hazards that will be resolved via forwarding? Which dependencies are data hazards that will cause a stall?

```
add $3, $4, $2
sub $5, $3, $1
lw  $6, 200($3)
add $7, $3, $6
```

6.5 [5] <§6.1> For More Practice: Delayed Branches

6.6 [10] <§6.2> Using Figure 6.22 on page 400 as a guide, use colored pens or markers to show which portions of the datapath are active and which are inactive in each of the five stages of the `sw` instruction. We suggest that you use five photocopies of Figure 6.22 to answer this exercise. (We hereby grant you permission to violate the Copyright Protection Act in doing the exercises in Chapters 5 and 6!) Be sure to include a legend to explain your color scheme.

6.7 [5] <§6.2> For More Practice: Understanding Pipelines by Drawing Them

6.8 [5] <§6.2> For More Practice: Understanding Pipelines by Drawing Them

6.9 [15] <§6.2> For More Practice: Understanding Pipelines by Drawing Them

6.10 [5] <§6.2> For More Practice: Pipeline Registers

6.11 [15] <§§4.8, 6.2> For More Practice: Pipelining Floating Point

6.12 [15] <§6.3> Figure 6.37 on page 417 and Figure 6.35 on page 415 are two styles of drawing pipelines. To make sure you understand the relationship between these two styles, draw the information in Figures 6.31 through 6.35 on pages 410 through 415 using the style of Figure 6.37 on page 417. Highlight the active portions of the data paths in the figure.

6.13 [20] <§6.3> Figure 6.14.10 is similar to Figure 6.14.7 on page 6.14-9 in the **For More Practice** section, but the instructions are unidentified. Determine as much as you can about the five instructions in the five pipeline stages. If you cannot fill in a field of an instruction, state why. For some fields it will be easier to decode the machine instructions into assembly language, using Figure 3.18 on page 205 and Figure A.10.2 on page A-50 as references. For other fields it will be easier to look at the values of the control signals, using Figures 6.26 through 6.28 on pages 403 and 405 as references. You may need to carefully examine Figures 6.14.5 through 6.14.9 to understand how collections of control values are presented (i.e., the leftmost bit in one cycle will become the uppermost bit in another cycle). For example, the EX control value for the subtract instruction, 1100, computed during the ID stage of cycle 3 in Figure 6.14.6, becomes three separate values specifying RegDst (1), ALUOp (10), and ALUSrc (0) in cycle 4.

6.14 [40] <§6.3> The following piece of code is executed using the pipeline shown in Figure 6.30 on page 409:

```
lw  $5, 40($2)
add $6, $3, $2
or  $7, $2, $1
and $8, $4, $3
sub $9, $2, $1
```

At cycle 5, right before the instructions are executed, the processor state is as follows:

a. The PC has the value 100_{ten}, the address of the `sub_instruction`.

b. Every register has the initial value 10_{ten} plus the register number (e.g., register $8 has the initial value 18_{ten}).

c. Every memory word accessed as data has the initial value 1000_{ten} plus the byte address of the word (e.g., Memory[8] has the initial value 1008_{ten}).

Determine the value of every field in the four pipeline registers in cycle 5.

6.15 [20] <§6.3> For More Practice: Labeling Pipeline Diagrams with Control

6.16 [20] <§6.4> For More Practice: Illustrating Diagrams with Forwarding

6.17 [5] <§§6.4, 6.5> Consider executing the following code on the pipelined datapath of Figure 6.36 on page 416:

```
add    $2,  $3,  $1
sub    $4,  $3,  $5
add    $5,  $3,  $7
add    $7,  $6,  $1
add    $8,  $2,  $6
```

At the end of the fifth cycle of execution, which registers are being read and which register will be written?

6.18 [5] <§§6.4, 6.5> With regard to the program in Exercise 6.17, explain what the forwarding unit is doing during the fifth cycle of execution. If any comparisons are being made, mention them.

6.19 [5] <§§6.4, 6.5> With regard to the program in Exercise 6.17, explain what the hazard detection unit is doing during the fifth cycle of execution. If any comparisons are being made, mention them.

6.20 [20] <§§6.4, 6.5> For More Practice: Forwarding in Memory

6.21 [5] <§6.5> We have a program of 10^3 instructions in the format of "`lw`, `add`, `lw`, `add`, ..." The `add` instruction depends (and only depends) on the `lw` instruction right before it. The `lw` instruction also depends (and only depends) on the `add` instruction right before it. If the program is executed on the pipelined datapath of Figure 6.36 on page 416:

a. What would be the actual CPI?

b. Without forwarding, what would be the actual CPI?

6.22 [5] <§§6.4, 6.5> Consider executing the following code on the pipelined datapath of Figure 6.36 on page 416:

```
lw   $4, 100($2)
sub  $6, $4, $3
add  $2, $3, $5
```

How many cycles will it take to execute this code? Draw a diagram like that of Figure 6.34 on page 414 that illustrates the dependencies that need to be resolved, and provide another diagram like that of Figure 6.35 on page 415 that illustrates how the code will actually be executed (incorporating any stalls or forwarding) so as to resolve the identified problems.

6.23 [15] <§6.5> List all the inputs and outputs of the forwarding unit in Figure 6.36 on page 416. Give the names, the number of bits, and brief usage for each input and output.

6.24 [20] <§6.5> For More Practice: Illustrating Diagrams with Forwarding and Stalls

6.25 [20] <§6.5> For More Practice: Impact on Forwarding of Moving It to ID Stage

6.26 [15] <§§6.2–6.5> For More Practice: Impact of Memory Addressing Mode on Pipeline

6.27 [10] <§§6.2–6.5> For More Practice: Impact of Arithmetic Operations with Memory Operands on Pipeline

6.28 [30] <§6.5, Appendix C> For More Practice: Forwarding Unit Hardware Design