EE557--FALL 1999

MIDTERM 1

Closed books, closed notes

GRADING POLICY: The front page of your exam shows your total numerical score out of 75. The highest numerical score was 55/75.

Your final grade (out of 25) was computed as follows:

\[
\text{Final\_grade(out\ of\ 25)} = \text{Total\_score(out\ of\ 75)} \times \frac{25}{55}.
\]

The highest grade was 25/25 and the lowest grade was 1.4/25.

Average was 11.8/25 and median was 11.2/25.
QUESTION 1 (Performance evaluation) 10 points

Consider two different implementations, M1 and M2, of the same instruction set. M1 has a clock rate of 400MHz, and M2 has a clock rate of 200MHz. The average number of cycles for each instruction class on M1 and M2 is given in the following table:

<table>
<thead>
<tr>
<th>Class</th>
<th>CPI on M1</th>
<th>CPI on M2</th>
<th>C1 usage</th>
<th>C2 usage</th>
<th>Third party usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>2</td>
<td>30%</td>
<td>30%</td>
<td>50%</td>
</tr>
<tr>
<td>B</td>
<td>6</td>
<td>4</td>
<td>50%</td>
<td>20%</td>
<td>30%</td>
</tr>
<tr>
<td>C</td>
<td>8</td>
<td>3</td>
<td>20%</td>
<td>50%</td>
<td>20%</td>
</tr>
</tbody>
</table>

The table also contains a summary of three different compilers using the instruction set. C1 is a compiler produced by the makers of M1, C2 is a compiler produced by the makers of M2, and the other compiler is a third-party product. Assume that each compiler uses the same number of instructions for a given program but that the instruction mix is as described in the table.

1. Using C1 on both M1 and M2, how much faster can the makers of M1 claim that M1 is compared to M2?

Answer: M1 is 1.103 times as fast as M2 because

\[
\text{CPI(M1)} = 0.3\times4 + 0.5\times6 + 0.2\times8 = 5.8 \\
\text{CPI(M2)} = 0.3\times2 + 0.5\times4 + 0.2\times3 = 3.2 \\
\text{SP(M1/M2)} = \frac{\text{CPI(M1)}}{\text{CPI(M2)}} \times \frac{\text{CL(M1)}}{\text{CL(M2)}} = \frac{3.2}{5.8} \times \frac{2}{2} = 1.10344
\]

2. Using C2 on both M2 and M1, how much faster can the makers of M2 claim that M2 is compared to M1?

Answer: M2 is 1.103 times as fast as M1 because

\[
\text{CPI(M1)} = 6.4; \quad \text{CPI(M2)} = 2.9 \text{ and} \\
\text{SP(M2/M1)} = \frac{6.4}{2.9} \times 1/2 = 1.103
\]

3. If you purchase M1, which compiler would you use?

Answer: Third-party because

\[
\text{CPI(C1)} = 5.8; \quad \text{CPI(C2)} = 6.4; \text{ and CPI(3rd party)} = 0.5\times4 + 0.3\times6 + 0.2\times8 = 5.4
\]
4. If you purchase M2, which compiler would you use?
Answer: Third-party because

\[ \text{CPI(C1)}=3.2; \text{CPI(C2)}=2.9 \text{ and CPI(3rd party)}= \frac{.5 \times 2 + .3 \times 4 + .2 \times 3}{2} = 2.8. \]

5. Which machine would you purchase given the choice of compiler, if we assume that all other criteria are identical, including costs?

Answer: M1 because with the 3rd party compiler,

\[ \text{SP(M1/M2)} = \frac{2.8}{5.4 \times 2} > 1 \]
QUESTION 2. Single Cycle CPU (10 points)

The data path in Figure 1 supports LW, SW, BEQ, and R-format instructions. We wish to add one instruction “ADD_Memory” (ADDM), which adds the content of a register to a memory location and stores the result in a register.

ADDM rs, rt, rd

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>25</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>001001</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>XXX</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description:** The word at memory location (rt) is added to the content of register rs and the result is stored in register rd.

1. (5pts) Draw the datapath modifications on Figure 1. Explain below. Make sure that all instructions still execute in one cycle.

Answer: We need one bus and one mux to route the value in rt to the memory address port. Also, the output of the data memory must be routed to the lower side of the ALU. See updated Figure 1.

2. (2pts) Specify the new control points (show them on Figure 1 and explain their settings here):

Answer: Mem@: selects the memory address in the data memory

Mem@="0" selects the output of the ALU; Mem@="1" selects the 2nd read data port (R2) of the register file.

ALUSrc may now take 3 values and is encoded with 2 bits instead of 1.

ALUSrc=00 selects R2; ALUSrc=01 selects the offset; and ALUSrc=10 selects data memory output.

3. (3pts) Give the values of all control points (0,1 or X) for the new instruction (except ALUOp) (including the new control points):

Answer: RegWrite=1; RegDest=1; ALUSrc=10; Branch=0; Mem@=1; MemRead=1; MemWrite=0; MemtoReg=0.
QUESTION 3 (20 pts)

Figure 2 shows a variation of the 5-stage pipeline for a cache based system with virtual memory. This new pipeline has 7 stages to accommodate the longer memory access times. Each instruction access takes two cycles: one to the instruction TLB (I-TLB in IF1) and one to the instruction cache (I-cache in IF2). Each load and store also takes two cycles: one through the data TLB (D-TLB in MEM1) and one through the data cache (D-cache in MEM2). The pipeline registers are named as in the 5-stage pipeline.

3.1. (10pts) Figure 2 shows the outputs of the forwarding and hazard detection units, with their name. On Figure 5, please draw the control and data buses which are the inputs to the forwarding unit and the hazard detection unit of this new pipeline.

Add the buses carrying the forwarded values.

Also number all inputs to the multiplexors drawn in bold.

Make sure that the inputs and buses are clearly labelled so that their functions are clear. ANSWER THIS QUESTION DIRECTLY ON FIGURE 2.

Answer:

See Figure 2.

3.2. (10pts) The logic equations for the FU (forwarding unit) and HDU (Hazard Detection Unit) of the 5-stage pipeline are in Figure 4 (also refer to figure 3). Use the same notation to specify the new FU (side A of the ALU only) and the new HDU for the 7-stage pipeline. Give the code on the next page.

Answer: See next page.
Answer:
Forwarding Unit (side A of ALU)

if (EX/MEM1.RegWrite and (EX/MEM1.RegisterRd = ID/EX.RegisterRs))
    then ForwardA=01 /*EX Hazard

else if (MEM1/MEM2.RegWrite and (MEM1/MEM2.RegisterRd = ID/EX.RegisterRs))
    then ForwardA = 10 /*MEM1 Hazard

else if (MEM2/WB.RegWrite and (MEM2/WB.RegisterRd = ID/EX.RegisterRs))
    then ForwardA = 11 /*MEM Hazard

else ForwardA = 00 /*No Hazard

Hazard Detection Unit

if (ID/EX.MemRead and
    ((ID/EX.RegisterRt = IF2/ID.RegisterRs) or (ID/EX.RegisterRt = IF2/ID.RegisterRt)))
    then {Annull=1; IF2/ID.CLK=0; IF1/IF2.CLK=0; PC.CLK=0} /*STALL

else if (EX/MEM1.MemRead and
    ((EX/MEM1.RegisterRd=IF/ID.RegisterRs) or (EX/MEM1.RegisterRd=IF/ID.RegisterRt)))
    then {Annull=1; IF2/ID.CLK=0; IF1/IF2.CLK=0; PC.CLK=0} /*STALL

else {Annull=0; IF2/ID.CLK=1; IF1/IF2.CLK=1; PC.CLK=1} /*NO STALL
QUESTION 4(15 pts) NOTE: This question may be tricky.

You are in charge of evaluating the benefits of executing conditional branches in the ID stage. The idea looks good. The branch address adder and a comparator can be included in the ID stage to compute the target address and the branch condition there. If branches are predicted not taken, only one cycle is lost on a taken branch. If you could only solve those nasty RAW hazards on the registers!!!

Your colleague has started to draft a possible solution to these RAW hazards and seems to have given up. In Figure 5, you can see that he plans to forward values from the EX and the MEM stages to the ID stages where they can be selected instead of the registers as inputs to the comparator.

Questions.

1. (2pt or 15pts) Do you think that the design as sketched in Figure 5 is really feasible, i.e. is it really possible to forward from the output of the ALU or of the Memory rather than from a pipeline register? The danger here is that signals from different instructions in different stages may enter a race.

Answer: YES.

If you answer NO to this question, then propose another (possibly lower performance but correct) way to solve the hazard. Please skip 2, 3, and 4 below and explain your idea on the next page.

If you answer YES, then proceed to questions below.

2. (4pts) Does the design sketch exploit all the forwarding opportunities? If it doesn’t, which one(s) should be added?

Answer:

No. The value from the data path coming out of the ALU output in the MEM stage should also be forwarded.

3. (4pts) Once all opportunities for forwarding have been exhausted, do you still have RAW hazards to solve? Which ones and how do you plan to solve them?

Answer:

Yes. After a LOAD instruction, we must stall the machine for one clock if there is a RAW hazard with a Branch instruction in the ID stage.
4. (2 pt) What is the effect on the clock cycle time of the pipeline? Please explain.

Answer:

The cycle time must now be larger than $T_{ALU}+T_{COMP}$ or than $T_{MEM}+T_{COMP}$. This will most probably increase the clock cycle time.

5. (3pts) It is well known that codes such as the following are extremely rare:

\[ \text{LW R1,0(R5)} \]
\[ \text{BEQ R1, R2, L1} \]

Knowing this, is there a way to change the design in order to improve the clock rate? Please explain.

Answer:

Yes. We could stall the processor for one more clock when the BEQ is preceded by a LW with a RAW hazard. This would allow us to remove the forwarding path between the output of the data memory and the ID stage. Thus the cycle time of the CPU must now be greater than $T_{ALU}+T_{COMP}$. Provided $T_{MEM} > T_{ALU}+T_{COMP}$, the cycle time is unchanged. This is the way to go since we need to optimize for the common case.
QUESTION 5 (20pts)

Figure 6 shows a DLX floating point pipeline with multiple execution units. The address (integer) unit takes one clock. The FP adder and FP multiplier are fully pipelined and take 3 and 6 clocks respectively.

1. (4pts) Please give the latency of some operations by filling out the following table

Answer:

<table>
<thead>
<tr>
<th>Table 2.</th>
<th>Latency (Clocks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD followed by ADDD</td>
<td>1</td>
</tr>
<tr>
<td>MULTIPLY followed by LOAD</td>
<td>0</td>
</tr>
<tr>
<td>LOAD followed by STORE</td>
<td>0</td>
</tr>
<tr>
<td>ADDD followed by STORE</td>
<td>1</td>
</tr>
<tr>
<td>MULTIPLY followed by STORE</td>
<td>4</td>
</tr>
<tr>
<td>MULTIPLY followed by MULTIPLY</td>
<td>5</td>
</tr>
<tr>
<td>ADDD followed by ADDD</td>
<td>2</td>
</tr>
</tbody>
</table>

2. (10pts) Consider the following program.

LOOP

LD F0,1000(R1)
LD F2,2000(R1)
MULTD F2,F0,F2
ADDD F4,F4,F2
SUBI R1,8
BNE R1,R0,LOOP

The branch is executed in the ID stage and forwarding is applied as in question 3. The branch is delayed by one instruction. In the following table please show the timing diagram for this loop by indicating the stage in each clock for each instruction in the table. Structural hazards on the write port of the register file are handled by issuing instructions only if they don’t conflict with previous instructions at the register file. This is done in the ID stage with a shift register.

Answer:

<table>
<thead>
<tr>
<th>Table 3.</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>LD F0,1000(R1)</td>
</tr>
<tr>
<td>LD F2,2000(R1)</td>
</tr>
<tr>
<td>MULTD F2,F0,F2</td>
</tr>
<tr>
<td>ADDD F4,F4,F2</td>
</tr>
<tr>
<td>SUBI R1,8</td>
</tr>
<tr>
<td>BNZ R1,LOOP</td>
</tr>
<tr>
<td>LD F0,1000(R1)</td>
</tr>
</tbody>
</table>
3. (1pts) Assuming that the loop is executed an infinite number of times, what is the time taken per iteration of the loop.

Answer:
TIME = 13 Clocks per iteration.

4. (5pts) Knowing that the branch is delayed by one, the compiler can schedule the code to improve performance. (No loop unrolling; just code scheduling within each iteration). Please show the new code with the stall. In this answer, ignore structural hazards. Show the stalls on the code. Estimate the time to execute one iteration of the loop and the speedup over the original code.

Answer:

```
LOOP:  
LD  F0,1000(R1)
LD  F2,2000(R1)
SUBI R1,8
MULTD F2,F0,F2
........
........
........
........
BNE R1,R0,LOOP
ADDD F4,F4,F2
```

TIME = 10 Clocks per iteration;
SPEEDUP = 1.3.
Figure 1. Simplified single cycle datapath
Figure 2 7-stage pipeline
Figure 3 (Simplified 5-stage pipeline)
Figure 4 CONDITIONS FOR FORWARDING AND HAZARD DETECTION IN THE 5-STAGE PIPELINE (refer to Figure 3)

CONDITION FOR FORWARDING:
(for side A of the ALU)
if (EX/MEM.RegWrite and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
   then ForwardA = 10 /EX Hazard
else if (MEM/WB.RegWrite and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
   then ForwardA = 01 /*MEM Hazard
else ForwardA = 00 /*No Hazard

The equations for side B of the ALU are similar.

CONDITION FOR STALLING
if (ID/EX.MemRead and
   ((ID/EX.RegisterRt = IF/ID.RegisterRs)
   or (ID/EX.RegisterRt = IF/ID.RegisterRt))
then {Annull=1; IF/ID.CLK=0; PC.CLK=0} /*STALL
else {Annull=0; IF/ID.CLK=1; PC.CLK=1} /*NO STALL
Figure 5 Proposed solution to the RAW hazard on branch operands
Figure 6. DLX pipeline with multiple execution units.