Lecture: Review Session

• Topics: load-store queue wrap-up, first half recap
Problem 2

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R1 ← [R2]</td>
<td>3</td>
<td></td>
<td>abcd</td>
</tr>
<tr>
<td>LD</td>
<td>R3 ← [R4]</td>
<td>6</td>
<td></td>
<td>adde</td>
</tr>
<tr>
<td>ST</td>
<td>R5 → [R6]</td>
<td>4</td>
<td>7</td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R7 ← [R8]</td>
<td>2</td>
<td></td>
<td>abce</td>
</tr>
<tr>
<td>ST</td>
<td>R9 → [R10]</td>
<td>8</td>
<td>3</td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R11 ← [R12]</td>
<td>1</td>
<td></td>
<td>abba</td>
</tr>
</tbody>
</table>
Problem 2

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R1 ← [R2]</td>
<td>3</td>
<td>abcd</td>
<td>4</td>
</tr>
<tr>
<td>LD</td>
<td>R3 ← [R4]</td>
<td>6</td>
<td>adde</td>
<td>7</td>
</tr>
<tr>
<td>ST</td>
<td>R5 → [R6]</td>
<td>4 7</td>
<td>abba</td>
<td>5</td>
</tr>
<tr>
<td>LD</td>
<td>R7 ← [R8]</td>
<td>2</td>
<td>abce</td>
<td>3</td>
</tr>
<tr>
<td>ST</td>
<td>R9 → [R10]</td>
<td>8 3</td>
<td>abba</td>
<td>9</td>
</tr>
<tr>
<td>LD</td>
<td>R11 ← [R12]</td>
<td>1</td>
<td>abba</td>
<td>2</td>
</tr>
</tbody>
</table>
Problem 3

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R1 ← [R2]</td>
<td>3</td>
<td>abcd</td>
<td></td>
</tr>
<tr>
<td>LD</td>
<td>R3 ← [R4]</td>
<td>6</td>
<td>adde</td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td>R5 → [R6]</td>
<td>5</td>
<td>7</td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R7 ← [R8]</td>
<td>2</td>
<td>abce</td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td>R9 → [R10]</td>
<td>1</td>
<td>4</td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R11 ← [R12]</td>
<td>2</td>
<td>abba</td>
<td></td>
</tr>
</tbody>
</table>
Problem 3

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R1 ← [R2]</td>
<td>3</td>
<td>abcd</td>
<td>4</td>
</tr>
<tr>
<td>LD</td>
<td>R3 ← [R4]</td>
<td>6</td>
<td>adde</td>
<td>7</td>
</tr>
<tr>
<td>ST</td>
<td>R5 → [R6]</td>
<td>5</td>
<td>7</td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R7 ← [R8]</td>
<td>2</td>
<td>abce</td>
<td>3</td>
</tr>
<tr>
<td>ST</td>
<td>R9 → [R10]</td>
<td>1</td>
<td>4</td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R11 ← [R12]</td>
<td>2</td>
<td>abba</td>
<td>3</td>
</tr>
</tbody>
</table>
Problem 4

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume memory dependence prediction.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R1</td>
<td>[R2]</td>
<td>3</td>
<td>abcd</td>
</tr>
<tr>
<td>LD</td>
<td>R3</td>
<td>[R4]</td>
<td>6</td>
<td>adde</td>
</tr>
<tr>
<td>ST</td>
<td>R5</td>
<td>[R6]</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>LD</td>
<td>R7</td>
<td>[R8]</td>
<td>2</td>
<td>abce</td>
</tr>
<tr>
<td>ST</td>
<td>R9</td>
<td>[R10]</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>LD</td>
<td>R11</td>
<td>[R12]</td>
<td>1</td>
<td>abba</td>
</tr>
</tbody>
</table>
### Problem 4

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume memory dependence prediction.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD R1</td>
<td>3</td>
<td>[R2]</td>
<td>abcd</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>LD R3</td>
<td>6</td>
<td>[R4]</td>
<td>adde</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>ST R5</td>
<td>4</td>
<td>[R6]</td>
<td>abba</td>
<td>5</td>
<td>commit</td>
</tr>
<tr>
<td>LD R7</td>
<td>2</td>
<td>[R8]</td>
<td>abce</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>ST R9</td>
<td>8</td>
<td>[R10]</td>
<td>abba</td>
<td>9</td>
<td>commit</td>
</tr>
<tr>
<td>LD R11</td>
<td>1</td>
<td>[R12]</td>
<td>abba</td>
<td>2</td>
<td>3/10</td>
</tr>
</tbody>
</table>
Solution:

<table>
<thead>
<tr>
<th>LD/ST</th>
<th>The register for the address calculation is made available</th>
<th>The register that must be stored into memory is made available</th>
<th>The calculated effective addr</th>
<th>Addr calculation happens</th>
<th>Data memory accessed</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>5</td>
<td>-</td>
<td>abcd</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>ST</td>
<td>7</td>
<td>4</td>
<td>abbb</td>
<td>8</td>
<td>at commit</td>
</tr>
<tr>
<td>LD</td>
<td>1</td>
<td>-</td>
<td>abbb</td>
<td>2</td>
<td>3/9</td>
</tr>
<tr>
<td>LD</td>
<td>3</td>
<td>-</td>
<td>abcd</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>ST</td>
<td>12</td>
<td>17</td>
<td>abbb</td>
<td>13</td>
<td>at commit</td>
</tr>
<tr>
<td>LD</td>
<td>2</td>
<td>-</td>
<td>abbb</td>
<td>3</td>
<td>4/9/18</td>
</tr>
</tbody>
</table>
### OOO Example

<table>
<thead>
<tr>
<th>Original code</th>
<th>Renamed code</th>
<th>InQ</th>
<th>Iss</th>
<th>Comp</th>
<th>Comm</th>
<th>Prev Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD R1, R2, R3</td>
<td>ADD P33, P2, P3</td>
<td>i</td>
<td>i+1</td>
<td>i+6</td>
<td>i+6</td>
<td>P1</td>
</tr>
<tr>
<td>LD R2, 8(R1)</td>
<td>LD P34, 8(P33)</td>
<td>i</td>
<td>i+2</td>
<td>i+8</td>
<td>i+8</td>
<td>P2</td>
</tr>
<tr>
<td>ADD R2, R2, 8</td>
<td>ADD P35, P34, 8</td>
<td>i</td>
<td>i+4</td>
<td>i+9</td>
<td>i+9</td>
<td>P34</td>
</tr>
<tr>
<td>ST R1, (R3)</td>
<td>ST P33, (P3)</td>
<td>i</td>
<td>i+2</td>
<td>i+8</td>
<td>i+9</td>
<td></td>
</tr>
<tr>
<td>SUB R1, R1, R5</td>
<td>SUB P36, P33, P5</td>
<td>i+1</td>
<td>i+2</td>
<td>i+7</td>
<td>i+9</td>
<td>P33</td>
</tr>
<tr>
<td>LD R1, 8(R2)</td>
<td>LD P1, 8(P35)</td>
<td>i+7</td>
<td>i+8</td>
<td>i+14</td>
<td>i+14</td>
<td>P36</td>
</tr>
<tr>
<td>ADD R1, R1, R2</td>
<td>ADD P2, P1, P35</td>
<td>i+9</td>
<td>i+10</td>
<td>i+15</td>
<td>i+15</td>
<td>P1</td>
</tr>
</tbody>
</table>
Problem 3

- Processor-A at 3 GHz consumes 80 W of dynamic power and 20 W of static power. It completes a program in 20 seconds.
What is the energy consumption if I scale frequency down by 20%?

What is the energy consumption if I scale frequency and voltage down by 20%?
**Problem 3**

- Processor-A at 3 GHz consumes 80 W of dynamic power and 20 W of static power. It completes a program in 20 seconds.

What is the energy consumption if I scale frequency down by 20%?

- New dynamic power = 64W; New static power = 20W
- New execution time = 25 secs (assuming CPU-bound)
- Energy = 84 W x 25 secs = 2100 Joules

What is the energy consumption if I scale frequency and voltage down by 20%?

- New dynamic power = 41W; New static power = 16W;
- New exec time = 25 secs; Energy = 1425 Joules
Problem 4

• Consider 3 programs from a benchmark set. Assume that system-A is the reference machine. How does the performance of system-B compare against that of system-C (for all 3 metrics)?

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sys-A</td>
<td>5</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Sys-B</td>
<td>6</td>
<td>8</td>
<td>18</td>
</tr>
<tr>
<td>Sys-C</td>
<td>7</td>
<td>9</td>
<td>14</td>
</tr>
</tbody>
</table>

- Sum of execution times (AM)
- Sum of weighted execution times (AM)
- Geometric mean of execution times (GM)
Problem 4

- Consider 3 programs from a benchmark set. Assume that system-A is the reference machine. How does the performance of system-B compare against that of system-C (for all 3 metrics)?

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>S.E.T</th>
<th>S.W.E.T</th>
<th>GM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sys-A</td>
<td>5</td>
<td>10</td>
<td>20</td>
<td>35</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>Sys-B</td>
<td>6</td>
<td>8</td>
<td>18</td>
<td>32</td>
<td>2.9</td>
<td>9.5</td>
</tr>
<tr>
<td>Sys-C</td>
<td>7</td>
<td>9</td>
<td>14</td>
<td>30</td>
<td>3</td>
<td>9.6</td>
</tr>
</tbody>
</table>

- Relative to C, B provides a speedup of 1.03 (S.W.E.T) or 1.01 (GM) or 0.94 (S.E.T)
- Relative to C, B reduces execution time by 3.3% (S.W.E.T) or 1% (GM) or -6.7% (S.E.T)
Problem 6

• My new laptop has a clock speed that is 30% higher than the old laptop. I’m running the same binaries on both machines. Their IPCs are listed below. I run the binaries such that each binary gets an equal share of CPU time. What speedup is my new laptop providing?

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old-IPC</td>
<td>1.2</td>
<td>1.6</td>
<td>2.0</td>
</tr>
<tr>
<td>New-IPC</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
</tr>
</tbody>
</table>
Problem 6

- My new laptop has a clock speed that is 30% higher than the old laptop. I’m running the same binaries on both machines. Their IPCs are listed below. I run the binaries such that each binary gets an equal share of CPU time. What speedup is my new laptop providing?

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>AM</th>
<th>GM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old-IPC</td>
<td>1.2</td>
<td>1.6</td>
<td>2.0</td>
<td>1.6</td>
<td>1.57</td>
</tr>
<tr>
<td>New-IPC</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
</tr>
</tbody>
</table>

AM of IPCs is the right measure. Could have also used GM. Speedup with AM would be 1.3.
Problem 2

• An unpipelined processor takes 5 ns to work on one instruction. It then takes 0.2 ns to latch its results into latches. I was able to convert the circuits into 5 sequential pipeline stages. The stages have the following lengths: 1ns; 0.6ns; 1.2ns; 1.4ns; 0.8ns. Answer the following, assuming that there are no stalls in the pipeline.

- What is the cycle time in the new processor?
- What is the clock speed?
- What is the IPC?
- How long does it take to finish one instr?
- What is the speedup from pipelining?
- What is the max speedup from pipelining?
Problem 2

- An unpipelined processor takes 5 ns to work on one instruction. It then takes 0.2 ns to latch its results into latches. I was able to convert the circuits into 5 sequential pipeline stages. The stages have the following lengths: 1ns; 0.6ns; 1.2ns; 1.4ns; 0.8ns. Answer the following, assuming that there are no stalls in the pipeline.

- What is the cycle time in the new processor? 1.6ns
- What is the clock speed? 625 MHz
- What is the IPC? 1
- How long does it take to finish one instr? 8ns
- What is the speedup from pipelining? $\frac{625}{192} = 3.26$
- What is the max speedup from pipelining? $\frac{5.2}{0.2} = 26$
Problem 8

• Consider this 8-stage pipeline (RR and RW take a full cycle)

IF  DE  RR  AL  AL  DM  DM  RW

• For the following pairs of instructions, how many stalls will the 2\textsuperscript{nd} instruction experience (with and without bypassing)?

- ADD R3 ← R1+R2
  ADD R5 ← R3+R4
- LD R2 ← [R1]
  ADD R4 ← R2+R3
- LD R2 ← [R1]
  SD R3 → [R2]
- LD R2 ← [R1]
  SD R2 → [R3]
Problem 8

- Consider this 8-stage pipeline (RR and RW take a full cycle)

| IF | DE | RR | AL | AL | DM | DM | RW |

- For the following pairs of instructions, how many stalls will the 2\textsuperscript{nd} instruction experience (with and without bypassing)?

  - ADD R3 ⟷ R1+R2
    ADD R5 ⟷ R3+R4
    without: 5  with: 1
  - LD R2 ⟷ [R1]
    ADD R4 ⟷ R2+R3
    without: 5  with: 3
  - LD R2 ⟷ [R1]
    SD R3 → [R2]
    without: 5  with: 3
  - LD R2 ⟷ [R1]
    SD R2 → [R3]
    without: 5  with: 1
Problem 1

• Consider a branch that is taken 80% of the time. On average, how many stalls are introduced for this branch for each approach below:
  ▪ Stall fetch until branch outcome is known
  ▪ Assume not-taken and squash if the branch is taken
  ▪ Assume a branch delay slot
    ○ You can’t find anything to put in the delay slot
    ○ An instr before the branch is put in the delay slot
    ○ An instr from the taken side is put in the delay slot
    ○ An instr from the not-taken side is put in the slot
Problem 1

- Consider a branch that is taken 80% of the time. On average, how many stalls are introduced for this branch for each approach below:
  - Stall fetch until branch outcome is known – 1
  - Assume not-taken and squash if the branch is taken – 0.8
  - Assume a branch delay slot
    - You can’t find anything to put in the delay slot – 1
    - An instr before the branch is put in the delay slot – 0
    - An instr from the taken side is put in the slot – 0.2
    - An instr from the not-taken side is put in the slot – 0.8
Problem 2

• Assume an unpipelined processor where it takes 5ns to go through the circuits and 0.1ns for the latch overhead. What is the throughput for 20-stage and 40-stage pipelines? Assume that the P.O.P and P.O.C in the unpipelined processor are separated by 2ns. Assume that half the instructions do not introduce a data hazard and half the instructions depend on their preceding instruction.
Problem 2

• Assume an unpipelined processor where it takes 5ns to go through the circuits and 0.1ns for the latch overhead. What is the throughput for 1-stage, 20-stage and 50-stage pipelines? Assume that the P.O.P and P.O.C in the unpipelined processor are separated by 2ns. Assume that half the instructions do not introduce a data hazard and half the instructions depend on their preceding instruction.

• 1-stage: 1 instr every 5.1ns
• 20-stage: first instr takes 0.35ns, the second takes 2.8ns
• 50-stage: first instr takes 0.2ns, the second takes 4ns
• Throughputs: 0.20 BIPS, 0.63 BIPS, and 0.48 BIPS
Problem 1

for (i=1000; i>0; i--)
    x[i] = y[i] * s;

Source code

Loop:    L.D         F0, 0(R1)          ; F0 = array element
          MUL.D    F4, F0, F2        ; multiply scalar
          S.D         F4, 0(R2)          ; store result
          DADDUI  R1, R1,# -8      ; decrement address pointer
          DADDUI  R2, R2,#-8       ; decrement address pointer
          BNE        R1, R3, Loop    ; branch if R1 != R3
          NOP

Assembly code

• How many cycles do the default and optimized schedules take?
Problem 1

for (i=1000; i>0; i--)
    x[i] = y[i] * s;

Source code

Loop:    L.D         F0, 0(R1)          ; F0 = array element
         MUL.D    F4, F0, F2        ; multiply scalar
         S.D         F4, 0(R2)          ; store result
         DADDUI  R1, R1,# -8      ; decrement address pointer
         DADDUI  R2, R2,#-8       ; decrement address pointer
         BNE        R1, R3, Loop    ; branch if R1 != R3
         NOP

Assembly code

Unoptimized: LD 1s  MUL 4s  SD  DA  DA  BNE 1s  -- 12 cycles
Optimized:  LD DA  MUL DA 2s  BNE  SD  -- 8 cycles

Degree 2: LD LD MUL MUL DA DA 1s SD BNE SD
Degree 3: LD LD LD MUL MUL MUL DA DA SD SD SD BNE SD
          -- 12 cyc/3 iterations
Source Code:
for (i=1000; i>0; i--) {
    w[i] = x[i] * w[i];
}

Assembly Code:
Loop:
L.D F1, 0(R1) // Get w[i]
L.D F2, 0(R2) // Get x[i]
MUL.D F1, F2, F1 // Multiply two numbers
S.D F1, 0(R1) // Store the result into w[i]
DADDUI R1, R1, #-8 // Decrement R1
DADDUI R2, R2, #-8 // Decrement R2
BNE R1, R3, Loop // Check if we've reached the end of the loop
NOP
Problem 3

```c
for (i=1000; i>0; i--)
    x[i] = y[i] * s;
```

Source code

```
Loop:     L.D         F0, 0(R1)          ; F0 = array element
          MUL.D    F4, F0, F2        ; multiply scalar
          S.D         F4, 0(R2)          ; store result
          DADDUI  R1, R1,# -8      ; decrement address pointer
          DADDUI  R2, R2,#-8       ; decrement address pointer
          BNE        R1, R3, Loop    ; branch if R1 != R3
          NOP
```

Assembly code

- How many unrolls does it take to avoid stalls in the superscalar pipeline?
Problem 3

for (i=1000; i>0; i--)
x[i] = y[i] * s;

Source code

<table>
<thead>
<tr>
<th>Loop:</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L.D</td>
<td>F0 = array element</td>
</tr>
<tr>
<td></td>
<td>MUL.D</td>
<td>multiply scalar</td>
</tr>
<tr>
<td></td>
<td>S.D</td>
<td>store result</td>
</tr>
<tr>
<td></td>
<td>DADDUI</td>
<td>decrement address pointer</td>
</tr>
<tr>
<td></td>
<td>DADDUI</td>
<td>decrement address pointer</td>
</tr>
<tr>
<td></td>
<td>BNE</td>
<td>branch if R1 != R3</td>
</tr>
<tr>
<td></td>
<td>NOP</td>
<td></td>
</tr>
</tbody>
</table>

Assembly code

LD -> any : 1 stall
FPMUL -> any: 5 stalls
FPMUL -> ST : 4 stalls
IntALU -> BR : 1 stall

• How many unrolls does it take to avoid stalls in the superscalar pipeline?
  7 unrolls. Could also make do with 5 if we moved up the DADDUIs.
Problem 2

- What is the storage requirement for a tournament predictor that uses the following structures:
  - a “selector” that has 4K entries and 2-bit counters
  - a “global” predictor that XORs 14 bits of branch PC with 14 bits of global history and uses 3-bit counters
  - a “local” predictor that uses an 8-bit index into L1, and produces a 12-bit index into L2 by XOR-ing branch PC and local history. The L2 uses 2-bit counters.
Problem 2

- What is the storage requirement for a tournament predictor that uses the following structures:
  - a “selector” that has 4K entries and 2-bit counters
  - a “global” predictor that XORs 14 bits of branch PC with 14 bits of global history and uses 3-bit counters
  - a “local” predictor that uses an 8-bit index into L1, and produces a 12-bit index into L2 by XOR-ing branch PC and local history. The L2 uses 2-bit counters.

Selector = 4K * 2b = 8 Kb
Global = 3b * 2^14 = 48 Kb
Local = (12b * 2^8) + (2b * 2^12) = 3 Kb + 8 Kb = 11 Kb
Total = 67 Kb
Problem 3

• For the code snippet below, estimate the steady-state bpred accuracies for the default PC+4 prediction, the 1-bit bimodal, 2-bit bimodal, global, and local predictors. Assume that the global/local preds use 5-bit histories.

```c
do {
    for (i=0; i<4; i++) {
        increment something
    }
    for (j=0; j<8; j++) {
        increment something
    }
    k++;
} while (k < some large number)
```
Problem 3

• For the code snippet below, estimate the steady-state bpred accuracies for the default PC+4 prediction, the 1-bit bimodal, 2-bit bimodal, global, and local predictors. Assume that the global/local preds use 5-bit histories.

do {
    for (i=0; i<4; i++) {
        increment something
    }
    for (j=0; j<8; j++) {
        increment something
    }
    k++;
} while (k < some large number)

PC+4: 2/13 = 15%
1b Bim: (2+6+1)/(4+8+1) = 9/13 = 69%
2b Bim: (3+7+1)/13 = 11/13 = 85%
Global: (4+7+1)/13 = 12/13 = 92%
Local: (4+7+1)/13 = 12/13 = 92%