Why are we here?

- Review important ideas not to be missed
- Go over some typical problems which you should know at least
- Not intended to cover everything – but at least stuff to get you going
- Plan to put this document on website
Some corrections to HW Solns given out

- HW #1, problem 8e
  - A MAL of 2 can be achieved, from a greedy cycle (1,3). This cycle loops indefinitely after a cycle time of 1 from start. Solution given incorrectly states (1,3,4) best cycle

- HW #2, problem 1a
  - Each loop takes exactly 20 cycles as opposed to given solution. Refer textbook problem A2’s solution for a similar explanation

- Some people need to collect back the homeworks they handed in. They are in Prof. Koren’s office
Important Points of Lectures 1-11

- Performance Characterization
  - Amdahl’s law – speedup, speedup........

- Use of reservation stations
  - Pipelining performance – MAL, Throughput

- Pipelining
  - Effect of different hazards
  - Ways to deal with these hazards
Amdahl’s Law and Related

- Speedup<sub>overall</sub> = ExTime(old)/ExTime(new)
  
  \[
  1 \over 1 - F_{enhanced} + \frac{F_{enhanced}}{Speedup_{enhanced}}
  \]

- CPU Time = IC \times CPI \times Clock Cycle Time

When comparing two cpu’s or programs with same instruction count,

\[
Speedup = \frac{CPI_A \cdot CT_A}{CPI_B \cdot CT_B}
\]
CPI Calculations

ALU – 40% freq, cycles 1
LOAD – 30% freq, cycles 2
STORE- 20% freq, cycles 1

- Avg. CPI = 1x0.4 + 2x0.30 + 0.2x1 = 1.2

Suppose your test program now has branches
BRANCH – 10% freq, cycles 2

- New Avg. CPI = 1.2 + 0.1x2 = 1.4

- Decrease in performance = 1.4/1.3
Stalls due to Hazards

- Most important for speedup calculations
- Every cycle stalled increases CPI, and hence increases execution time
- So make sure you know how to find number of stalls for data, structural and control hazards
Stalls are Everything

- If instructions share some functional unit, look out for structural hazards
- If instructions are dependent, find how many stalls would be required
  - Simple Pipeline
  - With forwarding
  - Scoreboard and Tomasulo
  - Scheduling code
- Find where branches are resolved, and how that affects number of stalls
  - Effect of prediction schemes
Sample Problem

☐ Suppose Branch Frequencies are:
  Conditional Branches -20%
  Jumps and Calls – 5%
60% of conditional branches are taken

☐ 4-deep pipeline with branch resolved at
  ■ end of second cycle for unconditional
  ■ end of third cycle of conditional

☐ Assume only first pipe stage independent of whether branch goes or not, ignore other stalls

☐ What is penalty of branch hazards, assuming we use predict ‘not-taken’?
Preliminaries

- Pipeline Speedup (ideal) = depth/(1+stalls) = 4/1 = 4
- Unconditional Branches – 5%
- Conditional Branches 20%
- Important things to look
  - Stalls due to unconditional branches
  - “ “ conditional taken branches
  - “ “ conditional not-taken branches
Stalls due to unconditional branches

<table>
<thead>
<tr>
<th>Instruction</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jump or Call</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
</tr>
<tr>
<td>i+1</td>
<td>IF</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td></td>
</tr>
<tr>
<td>i+2</td>
<td>X</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+3</td>
<td>X</td>
<td>IF</td>
<td>ID</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Thus we suffer 1 stall
Stalls due to conditional taken branches

<table>
<thead>
<tr>
<th>Instruction</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jump or Call</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
</tr>
<tr>
<td>i+1</td>
<td>IF</td>
<td>X</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td></td>
</tr>
<tr>
<td>i+2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>IF</td>
<td>ID</td>
<td></td>
</tr>
<tr>
<td>i+3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>IF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Thus we suffer 2 stalls
Stalls due to conditional non-taken branches

<table>
<thead>
<tr>
<th>Instruction</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jump or Call</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
</tr>
<tr>
<td>i+1</td>
<td>IF</td>
<td>X</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td></td>
</tr>
<tr>
<td>i+2</td>
<td>X</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+3</td>
<td>X</td>
<td>IF</td>
<td>ID</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Thus we suffer 1 stalls
Penalty due to branch hazards

- Thus Total stalls =
  \[0.05 \times 1 + 0.2 \times 0.6 \times 2 + 0.2 \times 0.4 \times 1 = 0.37\]

- Pipeline Speedup = \[\frac{4}{1 + 0.37} = 2.92\]

- Thus penalty in execution time = \[\frac{4}{2.92} = 1.37\]
Reservation Stations to model pipelines

- Should know how to model flow of instructions using reservation table
- Solving this table for MAL and T’put then should be simple
Sample Problem

The execution of the ADD instruction (with both operands and result in registers) in a certain pipelined processor takes 4 clock cycles. The processor has a single unified cache and a single register file. The register file is capable of executing, at any given clock cycle, either a single write operation or any number of read operations.
Operation Sequence

- In cycle 1, instruction is fetched
- In cycle 2, it is decoded and two operands are fetched
- In cycle 3, add operation is executed
- In cycle 4, result is stored in register file
- Assume a cache hit of 100%
Part A

- Write reservation table for ADD instruction

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF-ID</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EX</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF-W</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ALU</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Part B

What is Initial Collision Vector?

0 1 2 3

(1 0 1 0)

Draw State Diagram
Part C

- Find MAL associated with greedy cycle and calculate pipeline throughput

Possible Cycles: (1,3) or (3)

(1,3) is Greedy Cycle

MAL = (1+3)/2 = 2

T’put = 1/MAL

= 1 instruction in 2 cycles = 0.5
Sample Problem

- A given pipelined processor executes FP instructions. It has stages I and D taking 1 cycle each. Execution of FP ADD and MULTIPLY take 2 and 5 cycles respectively. Each Ex is followed by a singly clock cycle write back W into register file capable of supporting simultaneous reads but only single write per cycle.
Show all possible data and structural hazards

<table>
<thead>
<tr>
<th>S1: $F_1 = F_2 + F_3$</th>
<th>S4: $F_1 = F_1 \times F_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2: $F_4 = F_1 \times F_2$</td>
<td>S5: $F_2 = F_1 + F_6$</td>
</tr>
<tr>
<td>S3: $F_6 = F_4 + F_5$</td>
<td>S6: $F_4 = F_3 \times F_5$</td>
</tr>
</tbody>
</table>

Adder

RAW(F1)

S1

RAW(F1), WAR(F2)

S5

S6

RAW(WAW(F1))

Adder

S2

S3

S4

Finish it yourself!
Software Re-scheduling

- Assume there are multiple adders and multipliers now. i.e no more structural hazards. Show timing with forwarding possible
## Re-Scheduled Timing

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>I</td>
<td>D</td>
<td>A</td>
<td>A</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>I</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>W</td>
</tr>
<tr>
<td>S4</td>
<td>I</td>
<td>X</td>
<td>D</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>W</td>
</tr>
<tr>
<td>S3</td>
<td>X</td>
<td>I</td>
<td>D</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S6</td>
<td>I</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>D</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>W</td>
</tr>
<tr>
<td>S5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I</td>
</tr>
</tbody>
</table>

### Equations

- **S1**: $F_1 = F_2 + F_3$
- **S2**: $F_4 = F_1 \times F_2$
- **S3**: $F_6 = F_4 + F_5$
- **S4**: $F_1 = F_1 \times F_3$
- **S5**: $F_2 = F_1 + F_6$
- **S6**: $F_4 = F_3 \times F_5$
### Tomasulo Timing

|   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S1| I | D | A | A | W |   |   |   |   |   |   |   |   |   |   |   |   |
| S2| I | D | X | M | M | M | M | M | M | M | W |   |   |   |   |   |   |
| S3| I | D | X | X | X | X | X | X | X | X | A | A | W |   |   |   |   |
| S4| I | D | M | M | M | M | M | M | M | M | M | M | W |   |   |   |   |
| S5| I | D | X | X | X | X | X | X | X | X | A | A | W |   |   |   |   |
| S6| I | D | M | M | M | M | M | M | M | M | W |   |   |   |   |   |   |

**Equations:**

- **S1:** \( F_1 = F_2 + F_3 \)
- **S2:** \( F_4 = F_1 \times F_2 \)
- **S3:** \( F_6 = F_4 + F_5 \)
- **S4:** \( F_1 = F_1 \times F_3 \)
- **S5:** \( F_2 = F_1 + F_6 \)
- **S6:** \( F_4 = F_3 \times F_5 \)
Other topics to focus on

- Dynamic Branch prediction schemes
  - Branch history table
  - Correlation
  - Tournament Predictor
  - Branch Target Buffer

- I might have forgotten some topics – make sure you cover those
Tips to Study

- Do everything possible to understand everything clearly!
- Read and understand topic by topic, rather than read line by line of textbook
- Use internet resources
  - Plenty of similar courses – get practice
  - Solve as many problems as possible
- Solving Problems is Most Imp!