1. CPU performance

Suppose we have the following instruction mix and clock cycles per instruction.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Frequency</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU operations</td>
<td>30%</td>
<td>1</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>2</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>2</td>
</tr>
<tr>
<td>Branches</td>
<td>20%</td>
<td>3</td>
</tr>
<tr>
<td>Floating point operations</td>
<td>20%</td>
<td>5</td>
</tr>
</tbody>
</table>

(a) What is the overall CPI of this machine?

**Answer:**

\[
\text{CPI} = (0.3 \times 1) + (0.2 \times 2) + (0.1 \times 2) + (0.2 \times 3) + (0.2 \times 5)
\]

\[
= 0.3 + 0.4 + 0.2 + 0.6 + 1.0
\]

\[
= 2.5
\]

(b) If the CPU runs at 750MHz, what is the MIPS rating of this machine? For this question, count floating point operations in the MIPS rating.

**Answer:**

\[
\text{MIPS} = \frac{750 \text{MHZ}}{2.5 \text{ CPI}} = 300 \text{ MIPS}
\]

(c) Consider improving this computer’s performance by enhancing the speed of the floating point instructions. What is the best possible overall speedup that we could obtain?

**Answer:**

If we use Amdahl’s law, the best speedup we can get is:

\[
1/\left[\frac{1}{(1 - f)} + \frac{f}{n}\right] \implies \frac{1}{(1 - f)} \text{ as } n \text{ approaches } \infty
\]

In this case, \( f \) is the fraction of time devoted to floating point instructions, which is the same as the fraction of the CPI that comes from floating point instructions.

\[
f = \text{CPI for Floating Point instructions} / \text{Overall CPI}
\]

\[
= 1.0 / 2.5
\]

\[
= 0.4
\]

Therefore, the maximum speedup is:

\[
= \frac{1}{(1 - 0.4)}
\]

\[
\approx 1.67
\]
2. Speedup

Two enhancements, \( E_1 \) and \( E_2 \), with the following speedups are proposed for a new architecture:

\[
Speedup_1 = 10 \\
Speedup_2 = 5
\]

Only one of the enhancements is usable at any point in time (maybe because they use some of the same hardware).

(a) If \( E_1 \) can be used 20% of the time and \( E_2 \) can be used 10% of the time, what would be the overall speedup?

\[
\text{Answer:} \quad Speedup_{overall} = \frac{oldtime}{newtime} = \frac{1}{1 - f_1 - f_2 + f_1/sp_1 + f_2/sp_2} = \frac{1}{1 - .2 - .1 + .2/10 + .1/5} = \frac{1}{.7 + .02 + .02} = \frac{1}{.74} = 1.35
\]

(b) If the percentage of time that \( E_1 \) can be used decreased to 15%, what percentage of the time would the use of \( E_2 \) have to be to get the same overall speedup as in part (a)?

\[
\text{Answer:} \quad Speedup_{overall} = \frac{oldtime}{newtime} = 1.35 \\
\frac{1}{1 - f_1 - f_2 + f_1/sp_1 + f_2/sp_2} = 1.35 \\
\frac{1}{1 - .15 - f_2 + .15/10 + f_2/5} = 1.35 \\
\frac{1}{.865 - .8 * f_2} = 1.35 \\
1.35 * .865 - .8 * 1.35 * f_2 = 1 \\
.8 * 1.35 f_2 = 1.35 * .865 - 1 \\
f_2 = \frac{1.35 * .865 - 1}{.8 * 1.35} \approx .155 = 15.5\%
\]
(c) Suppose we are free to choose between E1 or E2, whenever we want (the percentages of time for using E1 or E2 can be varied as desired, but in total cannot be more than 100% of the time). What would be the maximum achievable overall speedup?

**Answer:**

Just use E1, because it provides higher speedup

\[
\text{Speedup}_{\text{overall}} = \frac{\text{oldtime}}{\text{newtime}} = \frac{1}{1 - f_1 + f_1/10} = \frac{1}{1 - 1 + 1/10} = 10
\]
3. MIPS ISA

(a) Suppose that \(a\) is stored in memory location 10, \(b\) is stored in memory location 20, and \(c\) is stored in memory location 30. Assume that all three variables are double word integers. Write MIPS assembly code equivalent to the following C code fragment:

\[
\begin{align*}
  a &= 4; \\
  c &= a + b;
\end{align*}
\]

Don’t worry about the exact syntax of the MIPS instructions (but get the addressing mode right). However, you must make your intent for each instruction clear. Remember that there is no load immediate MIPS instruction, to put a constant in a register.

**Answer:** Using only instructions shown in class:

\[
\begin{align*}
  \text{DADDI R1, R0, #4} & \quad \# \text{ load 4} \\
  \text{SD R1, 10(R0)} & \quad \# \text{ store into } a \\
  \text{LD R2, 20(R0)} & \quad \# \text{ load } b \\
  \text{DADD R3, R1, R2} & \quad \# a+b \\
  \text{SD R3, 30(R0)} & \quad \# \text{ store into } c
\end{align*}
\]

(b) Give 2 reasons why the MIPS architecture can get away with only 2 addressing modes, immediate and displacement.

**Answer:**

Because two other addressing modes can be emulated with displacement mode, register indirect (set offset to 0) and absolute (use R0 as the register), and because the other addressing modes aren’t needed very often so can be emulated in software with multiple MIPS instructions.
4. Basic pipelining

Consider the following MIPS code:

\[
\begin{align*}
I_1: & \text{ DADD R1, R5, R3} \\
I_2: & \text{ DADD R3, R1, R5} \\
I_3: & \text{ SD R5, 10(R1)} \\
I_4: & \text{ DADD R5, R1, R3}
\end{align*}
\]

Suppose we have the simple MIPS 5-stage pipeline from Appendix A in the book (and every instruction must go through all 5 stages).

(a) Fill in the first pipeline table showing execution of the instructions assuming that no forwarding is available, then fill out the second table assuming that forwarding is available in the pipeline wherever it is needed.

\[
\begin{array}{cccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\
I_1 & \text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} & & & & & & & & \\
I_2 & \text{IF} & \text{ST} & \text{ST} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} & & & & & & & \\
I_3 & & & & & & & & & & & & & \\
I_4 & & & & & & & & & & & & & \\
\end{array}
\]

With forwarding:

\[
\begin{array}{cccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\
I_1 & \text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} & & & & & & & & & \\
I_2 & \text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} & & & & & & & & & \\
I_3 & & & & & & & & & & & & & \\
I_4 & & & & & & & & & & & & & \\
\end{array}
\]

(b) Show and classify all the potential data hazards (RAW, WAR, WAW). Which of the potential RAW data hazards are eliminated because of forwarding?

Answer: $I_1 \rightarrow I_2$ (RAW, WAR)
$I_1 \rightarrow I_3$ (RAW)
$I_1 \rightarrow I_4$ (RAW, WAR)
$I_2 \rightarrow I_4$ (RAW, WAR)
$I_3 \rightarrow I_4$ (WAR)

The ($I_1 \rightarrow I_2$, $I_1 \rightarrow I_3$, and $I_2 \rightarrow I_4$) RAW dependences are eliminated by forwarding. The last RAW dependence, $I_1 \rightarrow I_4$, does not cause stalls even without forwarding, because its potential stall is hidden by the stalls for the other RAW dependences.