

Homework 1 Solution

1.3 This question further explores the effects of Amdahl's Law, but the data given in the question is in a form that cannot be directly applied to the general speedup formula.

- a. Because the information given does not allow direct application of Amdahl's Law we start from the definition of speedup:

$$\text{Speedup}_{\text{overall}} = \frac{\text{Time}_{\text{unenanced}}}{\text{Time}_{\text{enhanced}}}$$

The unenhanced time is the sum of the time that does not benefit from the 10 times faster speedup plus the time that does benefit, but before its reduction by the factor of 10. Thus,

$$\text{Time}_{\text{unenanced}} = 50\% \text{ Time}_{\text{enhanced}} + 10 \times 50\% \text{ Time}_{\text{enhanced}} = 5.5 \text{ Time}_{\text{enhanced}}$$

Substituting into the equation for Speedup yields

$$\text{Speedup}_{\text{overall}} = \frac{5.5 \text{ Time}_{\text{enhanced}}}{\text{Time}_{\text{enhanced}}} = 5.5$$

- b. Using Amdahl's Law, the given value of 10 for the enhancement factor, and the value for $\text{Speedup}_{\text{overall}}$ from part (a), we have

$$5.5 = \frac{\text{Fraction}_{\text{enhanced}}}{1 - \text{Fraction}_{\text{enhanced}} + \frac{\text{Fraction}_{\text{enhanced}}}{10}}$$

Solving shows that the enhancement can be applied 91% of the original time.

1.7 The exercise statement omits the value of the clock cycle time. We assume that the clock rate is the same for both processors and solve in terms of this unknown.

$$\text{a. } \text{IC}_{\text{RISC}} = \frac{\text{CPU time}_{\text{RISC}}}{\text{CPI}_{\text{RISC}} \times \text{CC}_{\text{RISC}}} = \frac{1.08 \text{ sec}}{10 \times \text{CC}} = 0.108 \times \text{CC instructions}$$

$$\text{IC}_{\text{emb}} = \frac{\text{CPU time}_{\text{emb}}}{\text{CPI}_{\text{emb}} \times \text{CC}_{\text{emb}}} = \frac{13.6 \text{ sec}}{6 \times \text{CC}} = 2.27 \times \text{CC instructions}$$

Relatively, there are 21 times more instructions executed by the embedded processor.

$$\text{b. } \text{MIPS}_{\text{RISC}} = \frac{\text{CC}_{\text{RISC}}}{\text{CPI}_{\text{RISC}} \times 10^6} = \frac{\text{CC}}{10 \times 10^6}$$

$$\text{MIPS}_{\text{emb}} = \frac{\text{CC}_{\text{emb}}}{\text{CPI}_{\text{emb}} \times 10^6} = \frac{\text{CC}}{6 \times 10^6}$$

The MIPS rating of the embedded processor will be a factor of $10/6 = 1.67$ times higher than the rating of the RISC version.

- c. The RISC processor performs the non-FP instructions plus 195,578 FP instructions. The embedded processor performs the same number of non-FP instructions as the RISC processor, but performs some larger number of instructions than 195,578 to compute the FP results using non-FP instructions only. The number of non-FP instructions is

$$\text{Number of non-FP instructions} = IC_{\text{RISC}} - 195578 = 0.108 \text{ CC} - 195578$$

Thus,

$$\begin{aligned} \text{Number instructions for FP}_{\text{emb}} &= IC_{\text{emb}} - \text{Number non-FP instructions} \\ &= 2.27 \text{ CC} - (0.108 \text{ CC} - 195578) \\ &= 2.16 \text{ CC} + 195578 \end{aligned}$$

Finally,

$$\begin{aligned} \text{Average number instr. for FP in software}_{\text{emb}} &= \frac{\text{Number instr. for FP}_{\text{emb}}}{\text{Number FP instr.}} \\ &= \frac{2.16 \text{ CC} + 195578}{195578} \end{aligned}$$

1.17 a. $MIPS_{\text{proc}} = 120 \times 10^6 = \frac{I + YF}{W}$

$$MIPS_{\text{proc/co}} = 80 \times 10^6 = \frac{I + F}{B}$$

b. $I = 120 \times 10^6 W - FY$
 $= (120 \times 10^6)(4) - (8 \times 10^6)(50)$
 $= 80 \times 10^6 \text{ instructions}$

c. $B = \frac{80 \times 10^6 + 8 \times 10^6}{80 \times 10^6} = 1.1 \text{ sec}$

d. $MFLOPS_{\text{proc/co}} = \frac{F}{B - \text{Time for integer instructions}}$
 $= \frac{F}{B - I/MIPS_{\text{proc/co}}}$
 $= \frac{8 \times 10^6}{1.1 - 80 \times 10^6 / 80 \times 10^6}$
 $= 80 \text{ MFLOPS}$

- e. The time for the processor alone is $W = 4 \text{ sec}$. The time for the processor/co-processor configuration is $B = 1.1 \text{ sec}$. While its MIPS rating is lower, the faster execution time belongs to the processor/co-processor combination. Your colleague's evaluation is correct.

2.6 This exercise analyzes the trade-off between adding a new instruction that decreases the number of instructions executed at the cost of increasing the clock cycle by 5%.

- a. Because the new design has a clock cycle time equal to 1.05 times the original clock cycle time, the new design must execute fewer instructions to achieve the same execution time. For the original design we have

$$\text{CPU Time}_{\text{old}} = \text{CPI}_{\text{old}} \times \text{CC}_{\text{old}} \times \text{IC}_{\text{old}}.$$

For the new design the equation is

$$\begin{aligned} \text{CPU Time}_{\text{new}} &= \text{CPI}_{\text{new}} \times \text{CC}_{\text{new}} \times \text{IC}_{\text{new}} \\ &= \text{CPI}_{\text{old}} \times (\text{CC}_{\text{old}} \times 1.05) \times (\text{IC}_{\text{old}} - R) \end{aligned}$$

where the CPI of the new design is the same as the original (per the exercise statement), the new clock cycle is 5% longer, and the new design executes R fewer instructions than the original design. To find out how many loads must be removed to match the performance of the original design set the above two equations equal and solve for R :

$$\begin{aligned} \text{CPI}_{\text{old}} \times \text{CC}_{\text{old}} \times \text{IC}_{\text{old}} &= \text{CPI}_{\text{old}} \times (\text{CC}_{\text{old}} \times 1.05) \times (\text{IC}_{\text{old}} - R) \\ R &= 0.048 \text{ IC}_{\text{old}} \end{aligned}$$

Thus the instruction count must decrease by 4.8% overall to achieve the same performance, and this 4.8% is comprised entirely of loads. Figure 2.32 shows that 25.1% of the gcc instruction mix is loads so $(4.8\%/25.1\%) = 19.0\%$ of the loads must be replaced by the new register-memory instruction format for the performance of the old and new designs to be the same. If more than 19% of the loads can be replaced then performance of the new design is better.

- b. Consider the code sequence

```
LOAD    R1,0(R1)
ADD     R1,R1,R1
```

The result written to R1 is $\text{MEM}[0+\text{R1}] + \text{MEM}[0+\text{R1}]$. However, the register-memory form of this code sequence is

```
ADD     R1,0(R1)
```

The result written to R1 in this case is $\text{R1} + \text{MEM}[0+\text{R1}]$ which is not the same unless the initial value in R1 just happens to be equal to the value at the memory location with address $[0+\text{R1}]$. In general the values will not be equal, and a compiler would not be able to use the new instruction form in this case.

2.12 This exercise provides an example of how a computer architect might go about analyzing the potential performance impact of a new architectural feature. It explores the effect of adding a new addressing mode that, in some cases, reduces the number of instructions required to perform an effective address calculation in support of a load or store instruction. The change allows for effective addresses that are the sum of two registers and a constant, as compared to the base instruction set capability of forming effective addresses as the sum of a single register and a constant. The first part of the exercise examines the change in program instruction count caused by the instruction set change. The second part evaluates the speedup achieved using the modification.

- a. First, we must understand exactly how the proposed instruction set modification changes the instruction count. Look again at the code fragment in the exercise statement.

```
ADD      R1,R1,R2      ; R1 = R1 + R2
LW       Rd,100(R1)    ; load from 100 + R1
```

The ADD instruction is used to compute an intermediate value of the effective address for the LW, then the LW instruction adds the final amount of 100 to produce the intended address. For the new MIPS with the index addressing mode the code fragment would be

```
LW       Rd,100(R1,R2) ; load from 100 + R1 + R2
```

In this code, the ADD has been folded into the load (or store) instruction. To make this possible some of the bits of the constant offset field of the original displacement load (or store) instruction must instead be used to encode the name of the second register. The remaining offset bits can encode a reduced range of offset values. This is one of the tradeoffs of the new addressing mode and is the reason that some ADD/LW sequences may not be able to use this new instruction form.

The exercise statement says that 10% of the displacement loads and stores in the original code can use the new form. From Figure 2.32, these loads and stores comprise 26% and 10% of the average instruction mix, respectively. Thus, assuming that MIPS executes IC_{original} instructions, then the ratio of instructions executed by the enhanced MIPS to the original MIPS is

$$\frac{IC_{\text{enhanced}}}{IC_{\text{original}}} = \frac{IC_{\text{original}} - IC_{\text{original}}[(10\% \times 26\%) + (10\% \times 10\%)]}{IC_{\text{original}}} = 0.964$$

The enhanced MIPS executes 3.6% fewer instructions than the original MIPS.

- b. While the new addressing mode reduces the total instruction count, the work done by the new instruction requires addition of three operands to compute the effective address. It is thus reasonable that this might require a slower clock cycle to allow effective address computation to complete. The exercise statement says that the increase is 5%. To determine which version of MIPS is faster, go back to our gold standard of performance, CPU time and use that in the speedup equation. One unspecified parameter of the CPU time equation is the difference in CPI, if any, between the version of MIPS. In the absence of information about CPI or that can be used to compute relative CPI for the two machines, we assume that CPI is unchanged. This is not an unreasonable assumption because the fraction of affected instructions, 3.6%, is relatively small. However, for a more precise analysis of the speedup offered by the enhanced machine, CPI should be examined more closely.

Considering all of the above gives the following equation for speedup.

$$\begin{aligned}
 \text{Speedup} &= \frac{\text{CPI}_{\text{original}} \times \text{CC}_{\text{original}} \times \text{IC}_{\text{original}}}{\text{CPI}_{\text{enhanced}} \times \text{CC}_{\text{enhanced}} \times \text{IC}_{\text{enhanced}}} \\
 &= \frac{\text{CPI}_{\text{original}} \times \text{CC}_{\text{original}} \times \text{IC}_{\text{original}}}{(\text{CPI}_{\text{original}})(1.05 \text{CC}_{\text{original}})(0.964 \text{IC}_{\text{original}})} \\
 &= 0.988
 \end{aligned}$$

As it turns out, even though the instruction count is lower for the enhanced MIPS, the increase in clock cycle time hurts! The enhanced MIPS is actually 1.2% slower than the original MIPS, implying that this architectural feature is not one to include in a future version of the machine.

There is an important lesson here for computer designers. In the same way that squeezing part of an elastic balloon fails to reduce its total volume because of bulging elsewhere, so to is improving one aspect of the CPU time equation often accompanied by worsening of another aspect. Candidate machine enhancements must be evaluated with respect to time and not just for their effect on one factor of CPU time.