Computer Architecture - tutorial 1 [TUTOR COPY]

Context, Objectives and Organization

This worksheet covers material from Lectures 1 (introduction to CA and technology trends), 2 (CPU performance equations) and 3 (the first half of the section on Instruction Set Architecture). The main goals of this tutorial is to give you some quantitative experience with the CPU performance equation, and to stimulate a discussion where your group can explore some issues related to modern instruction set design.

This tutorial consists of two activities. The first involves quantitative problem solving using exercises taken from the 2nd and 3rd editions of the H&P book. These can be done individually or in groups of two. The second main activity involves qualitative discussions in small groups followed by an exchange of ideas with the rest of the group.

E1: H&P (2/e) 1.6 p.61 – individual or groups of 2, 15 mins.

Problem

After graduating, you are asked to become the lead computer designer at Hyper Computers, Inc. Your study of usage of high-level language constructs suggests that procedure calls are one of the most expensive operations. You have invented a scheme that reduces the loads and stores normally associated with procedure calls and returns. The first thing you do is run some experiments with and without this optimization. Your experiments use the same state-of-the-art optimizing compiler that will be used with either version of the computer. These experiments reveal the following information:

- The clock rate of the unoptimized version is 5% higher.
- 30% of the instructions in the unoptimized version are loads or stores.
- The optimized version executes 2/3 as many loads and stores as the unoptimized version. For all other instructions the dynamic counts are unchanged.
- All instructions (including load and store) take one clock cycle.

Which is faster? Justify your decision quantitatively.

Solution

CPU performance equation: $CPUTime = IC * CPI * ClockTime$

We know:

$ClockTime_{unop} = 0.95 * ClockTime_{op}$
$IC_{ld/st,unop} = 0.3 * IC_{unop}$
$IC_{ld/st,op} = 0.67 * IC_{ld/st,unop}$
$IC_{others,unop} = IC_{others,op}$
$CPI = 1$
Thus:

$$\begin{aligned} CPUTime_{unop} &= IC_{unop} * ClockTime_{unop} = 0.95 IC_{unop} * ClockTime_{op} \\ CPUTime_{op} &= IC_{op} * ClockTime_{op} \end{aligned} \quad (1)$$

but,

$$IC_{op} = 0.7 * IC_{unop} + 0.3 * 0.67 * IC_{unop} = 0.9 * IC_{unop}$$

then,

$$CPUTime_{op} = 0.9 * IC_{unop} * ClockTime_{op} \quad (2)$$

Comparing 1 and 2 we see that the optimized version is faster.

$$Speedup = \frac{0.95 * IC_{unop} * ClockTime_{op}}{0.9 * IC_{unop} * ClockTime_{op}} = 1.06$$

E2: H&P (2/e) 2.6 p.164 – individual or groups of 2, 10 mins.

Problem

Several researchers have suggested that adding a register-memory addressing mode to a load-store machine might be useful. The idea is to replace sequences of:

```
LOAD   Rx,0(Rb)
ADD    Ry,Ry,Rx
```

by

```
ADD    Ry,0(Rb)
```

Assume this new instruction will cause the clock period of the CPU to increase by 5%. Use the instruction frequencies for the gcc benchmark on the load-store machine from Table 1. The new instruction affects only the clock cycle and not the CPI.

1. What percentage of the loads must be eliminated for the machine with the new instruction to have at least the same performance?

2. Show a situation in a multiple instruction sequence where a load of a register (Rx) followed immediately by a use of the same register (Rx) in an ADD instruction, could not be replaced by a single ADD instruction of the form proposed.

Instruction	**Frequency**
load	22.8%
store	14.3%
add	14.6%
sub	0.5%
mul	0.1%
div	0%
compare	12.4%
load imm	6.8%
cond. branch	11.5%
uncond. branch	1.3%
call	1.1%
return, jump ind.	1.5%
shift	6.2%
and	1.6%
or	4.2%
other (xor, not)	0.5%

Table 1: Instruction frequencies for gcc (cc1) (from H&P (2/e), Figure 2.26, p.105)

Solution

CPU performance equation: $CPUTime = IC * CPI * ClockTime$

We know:

$ClockTime_{op} = 1.05 * ClockTime_{unop}$

$CPI_{op} = CPI_{unop}$

a.

$$\begin{aligned} CPUTime_{op} &= CPUTime_{unop} \\ \Rightarrow IC_{op} * 1.05 * ClockTime_{unop} &= IC_{unop} * ClockTime_{unop} \\ \Rightarrow IC_{op} &= 0.95 * IC_{unop} \end{aligned} \tag{3}$$

but,

$$IC_{op} = IC_{unop} - R \tag{4}$$

where R is the number of instructions removed,
then combining 3 and 4,

$$\begin{aligned} IC_{unop} - R &= 0.95 * IC_{unop} \\ \Rightarrow R &= 0.05 * IC_{unop} \end{aligned} \tag{5}$$

but (from figure 2.32, pg. 138),

$$IC_{ld,unop} = 0.228 * IC_{unop} \tag{6}$$

combining 5 and 6 ,

$$\begin{aligned} R &= 0.05 * 4.39 * IC_{ld,unop} \\ \Rightarrow R &= 0.22 * IC_{ld,unop} \end{aligned}$$

b.

Consider the following code:

```
LOAD   R1, 0(R5)
ADD    R1, R1, R1
```

and that r1 has the initial value 47, r5 has a value of 1000 and that the data value in memory[1000] is 4. Then replacing the code with:

```
ADD    R1, 0(R5)
```

will give the incorrect result of 51, instead of the correct result 8. In other words, in situations in which Rx and Ry refer to the same register, the replacement will not be semantically equivalent.

D1: Discussion – in groups of 4, 15 mins.

In the early years of the RISC versus CISC dispute, the total number of different instructions and their variations in the IS was a common indication of the "simplicity" of an ISA (lesser the number, greater the simplicity). Modern RISC instruction sets contain almost as many instructions as old CISC instruction sets. Discuss whether modern "RISC" processors no longer RISC (as envisioned in the 80's). If they are still RISC, then what features in the instruction set best define the simplicity of an ISA? (e.g. memory access instructions, fixed and simple instruction encoding, register-oriented instructions, simple data types, etc?)

D2: Discussion – in groups of 4, 10 mins.

Even though the Intel x86 ISA is a clear example of a CISC ISA, modern implementations of it (e.g. Core and Xeon) use many RISC ideas: register-based micro-instructions, pipelining, simple branch micro-instructions, fixed lengthmicro-instructions, etc. Some say that, since at the low level the latest Intel processors *behave* like a RISC, they are RISC. Others say that, since at the software interface (compiler) they are *seen* as CISC, they are CISC. Discuss at what level ISA complexity should be measured. What are the implications of considering the ISA at each level? Are the latest Intel processors RISC?

Discussion To aid the discussion, introduce modern approaches for execution of CISC instructions: "cracking" into micro-ops, many physical registers to complement architectural registers, etc.

Vijay Nagarajan 2018. Thanks to Nigel Topham, Marcelo Cintra and Boris Grot.