

ECSE 425
Computer Organization and Architecture

Problem Set 2

Fall 2008

Question 1: Tradeoffs

The compiler for a given processor is enhanced by an optimization feature which has the effect of replacing half of the “fixed-point multiplication” instructions by two standard ALU instructions for each of the replaced fixed-point multiplication instructions.

Benchmarking indicates that 5% of all original instructions are fixed-point multiplies with a CPI of 4.0, 30% are load/stores with a CPI of 1.5, 45% are ALU instruction with a CPI of 1.2, and the rest have a CPI of 2. Evaluate the expected speedup provided by this compiler optimization.

Question 2: CPI

The dynamic instruction count is the number of instructions a processor actually executes (not the number of instructions in the object code produced by the compiler). Figure B.27 in the textbook gives the dynamic instruction mix for five programs from the SPECint2000 benchmark suite run on a MIPS processor. We can group instructions into categories. The average number of clock cycles for each category has been measured to be:

All ALU Instructions	1.4
Loads/Stores	1.6
Branches	2.8
Jumps	1.2
All other inst. (call, return, ...)	2.4

Use the averaged measurements from the figure to find the instruction mix in terms of the above categories. What is the effective CPI for the MIPS machine?

Question 3: MIPS64 machine code

- A) Write a MIPS64 code fragment to implement the following C code fragment, assuming that `i` and `c` are integer variables. Assembly language programs can be very hard to decipher, so make sure you add descriptive comments to every line of assembly code (i.e. “Add R2 and R3 and store in R1” is NOT a very descriptive comment for `DADD R1, R2, R3`. Describe what the purpose of the line is.)

To detect truncation, perhaps while building a pathname, something like the following might be used:

```
char *dir, *file, pname[MAXPATHLEN];
...
if (strncpy(pname, dir, sizeof(pname)) >= sizeof(pname))
    goto toolong;
if (strncat(pname, file, sizeof(pname)) >= sizeof(pname))
    goto toolong;
```

Since we know how many characters we copied the first time, we can speed things up a bit by using a copy instead of an append:

```
char *dir, *file, pname[MAXPATHLEN];
size_t n;
...
n = strncpy(pname, dir, sizeof(pname));
if (n >= sizeof(pname))
    goto toolong;
if (strncpy(pname + n, file, sizeof(pname) - n) >= sizeof(pname) - n)
    goto toolong;
```

However, one may question the validity of such optimizations, as they defeat the whole purpose of `strncpy()` and `strncat()`. As a matter of fact, the first version of this manual page got it wrong.

SEE ALSO
`snprintf(3)`, `strncat(3)`, `strncpy(3)`

HISTORY
`strncpy()` and `strncat()` functions first appeared in OpenBSD 2.4, and made their appearance in FreeBSD 3.3.

BSD

June 22, 1998

BSD

Question 4: ISA analysis

The purpose of this question is to get you to analyze an arbitrary ISA.

The PowerPC is an extremely successful architecture designed to meet the highly diverse needs of solutions ranging from desktop computer CPUs to high performance, highly integrated embedded processors. Skim through the first 2 chapters of the PowerPC Microprocessor Manual posted on myCourses.

In the following questions, ignore the AltiVec vector instructions.

After consulting Chapter 1, pages 1-32 – 1-59, and Chapter 2, pages 2-59 – 2-100, answer the following questions:

- What types of PowerPC registers does it have? (List them)
- What are the main instructions types that it includes? (e.g. floating point)
- What types of integer instructions are there?
- Does it have fixed or variable length instruction encoding?
- How big is the effective address?

- What sizes of memory operands are supported?
- Which instruction must operate on aligned operands?
- What are the addressing modes for loads/stores?
- How many bits does the special register for testing and branching have? What is it called?

Question 5: IC's and Mixes

The purpose of this question is to use `gcc` to see how the levels of compiler optimization can dramatically affect both the instruction count and the instruction mix for a given machine and a given code.

You will need to use a Sun SPARC processor to answer this question, for example, the ECE UNIX machines.

As a toy benchmark, we use a matrix multiply routine (call it `M.c`):

```
double A[20][20], B[20][20], C[20][20];

void mmult(){
    int i, j, k;
    double sum;
    for(i = 0; i < 20; i++) {
        for(j = 0; j < 20; j++) {
            for(k = 0, sum = 0.0; k < 20; k++) {
                sum += A[i][k] * B[k][j];
                C[i][j] = sum;
            }
        }
    }
}
```

The task is to find the instruction count and the instruction mix for the three nested loops. To do that, you will just perform a "hand count".

First, get `gcc` to produce human readable assembly code (`-S` option), for two optimization levels. Start with the highest.

```
gcc M.c -S -O3
```

At this point you end up with a file `M.s` that contains the corresponding assembly code. With the help of the assembly syntax manual for the SPARC machine (see WebCT), find out the sequence of instructions corresponding to these three nested loops (in a UNIX/SPARC/gcc environment, this code will start at label `.LL5` and ends at `.LLfe1`). Save this version of the assembly code, e.g.: "`mv M.s MO.s`".

Repeat the same as above with:

```
gcc M.c -S
```

This time, find out by yourself the instruction sequence corresponding to the three nested loops.

What are the instruction counts and mixes in the two cases? Make a table to show the instruction mix by grouping the types of instructions into categories (int ALU, fp ALU, int load/store, fp load/store, branches, jumps, moves, “no operation” (nop)). What was the effect of the optimization? What has the optimizer done to the code to make it more efficient (or less efficient if that is your answer). Were certain categories of instructions affected in the instruction mix? Which ones, how were they affected, and why?

Hint: Take into account the fact that the algorithm in M.c is iterative.