CS418	Advanced Computer Architecture	Spring 1995

Homework 3: Due 24 Jan

Reading

Tue	17	Jan	H&P 3.1, 3.2, 3.7, 3.8; skim 3.3–3.6
Thur	19	Jan	H&P 4.1, 4.5–4.7; skim 4.2–4.4

Problem 1 (2 Points)

Which of the following parameters should be defined by an instruction set architecture and which should be allowed to differ for different microprocessors implementing the same architecture? For each paramter, give a short (no more than 30 words) justification of your decision.

Feature	Arch Defined	Imp Defined
Number of bits per instruction		
Addressing modes		
Number of registers		
Number of cache levels		
Size of caches		
Size of main memory		
Size of virtual memory		
Latency for a floating-point divide		
Average CPI for all instructions		
Processor speed		
Pipelined vs. non-pipelined		
Superscalar vs. non-superscalar		
Big-endian vs little endian		
Number of pins on the chip		

Problem 2 (5 Points) (Based on H&P Exercise 3.8)

This exercise examines some of the issues related to the algorithms compilers use to perform register allocation algorithms and choose addressing modes. Throughout this problem we will examine the different ways of writing assembly code to access an element of an array of double precision floats. In particular, we will look at the code fragment for: `A[i] + x`.

Information about the program:

Variable	Type	Storage location
`A`	array of double precision floats	base address in `R1`
`i`	index into `A`	`R2`
`x`	double precision float	`R3`

Information about the microprocessor:

Speed: 100MHz
Memory is addressed in 8 bit bytes
Double precision floats are 64 bits
The data-bus between the core CPU and memory is 64 bits wide
CPI for all instructions: 1 + memory latency

Addressing mode	memory latency	total CPI
Indexed	4	5
Register indirect	3	5
Register-register	0	1

For the above microprocessor, there are two different approaches to writing the code to access `A[i]`:

1. use indexed addressing to access `A[i]`
2. compute the address of `A[i]`, store this address in a register (*e.g.* `R5`), and use register-indirect to access `A[i]`

For parts (a), (b), and (c) you will compare using different addressing modes for a compiler that stores the *addresses* of array elements in registers, and keeps the *values* of array elements in memory. (This decision makes sense for architectures that do not have very many registers.) For part (c) you will compare using different addressing modes for a compiler that stores the *addresses* and *values* of array elements in registers.

(a) (1 Point) Write the assembly code for `A[i] + x` using each of the two approaches

(b) (1 Point) How many references to `A[i]` must occur for a program compiled using method 2 (register-indirect) to be faster?

(c) (1 Point) Suppose, when using method 2, you run out of registers and have to save the address of `A[i]` on the stack and then restore it each time `A[i]` is accessed. If the stack is accessed using the indexed addressing mode, how many references to `A[i]` must occur for method 2 to be faster?

(d) (1 Point) The same company has a slightly different architecture that has more registers. For this architecture it makes sense to keep both *addresses* and *values* of operands in registers. Assuming that all there are sufficient registers to keep the value of `A[i]` in a register (*i.e.* you never need to save it on the stack), how many references must now occur to make method 2 faster?

(e) (1 Point) What conclusions can you make about the effect of architectures on compilers' algorithms for choosing addressing modes?

Problem 3 (4 Points) (Based on H&P Exercise 4.4)

When designing memory systems and the interfaces between the microprocessor core and memory, we need to know statistics such as the ratio of reads to writes for data accesses and the ratio of data accesses to instruction accesses. Use the average instruction-mix information for the DLX (Appendix C of H&P) to answer the following questions. (Ignore the size of datum, *e.g.* long ints vs. ints vs. single precision floats vs. double precision floats)

(a) (1 Point) Find the percentage of *all* accesses that are for *data*.

(b) (1 Point) Find the percentage of *data* accesses that are *reads*.

(c) (1 Point) Find the percentage of *all* accesses that are *reads*.

(d) (1 Point) The data for the DLX is fairly typical for a load/store machine. Briefly describe and justify how you think the numbers you calculated in parts (a), (b), and (c) compare to a register-memory architecture, such as the Intex x86.

Problem 4 (4 Points) (H&P Exercise 4.12)

Problem 5 (1 Point)

How much time did you spend on each of these problems? What was the educational value of each problem? How well did the lectures and text book prepare you for this homework? (Which concepts needed better explanation?)