

CS 573
Final Exam
May 14, 2004

The following exam is open book and open notes. You may feel free to use whatever additional reference material you wish, but **no electronic aids** are allowed. Please note the following instructions. There will be a ten point deduction for failure to comply with them:

- start each problem on a new sheet of paper
- write your social security number, but not your name, on each sheet of paper you turn in
- show your work whenever appropriate. There can be no partial credit unless I see how answers were arrived
- be succinct. You may lose points for facts that, while true, are not relevant to the question at hand

You have until 12:30 to finish the exam. The questions are equally weighted.

1. Several years ago, reading Usenet, I came across the following claim:

```
Subject: Re: Is RISC dead? (was: Re: K7's FP performance inches past P-III's)
Date: 1999/05/19
Author: Piercarlo Grandi <piercarl@Dial.PIPEx.com>
Well, my take here (and I said so in "comp.arch" quite a while ago) is that _architecture_ is
dead: with essentially infinite silicon/design budgets virtually any architecture, including
x86, can be made to perform; in other words only implementation matters.
```

Evaluate this claim. Is it the case that, today, the instruction set has become unimportant? In answering the question, consider the instruction set as the interface between the programmer's intent and the out-of-order execution engine executing it.

This one is really a judgement call, and I'll be looking much more carefully at your arguments than your final Yes or No.

There are a couple of different things to look at here. First, the statement is obviously false if we regard issues such as the word (both data and address) size. As applications need greater address spaces, these have to get bigger, and so at least that part of the word size remains important.

Second, the statement is also false if we find an instruction set whose semantics are so complex that (even with today's implementation budgets) a fast implementation isn't possible: this is probably the lesson to learn from the failure (so far) of Itanium.

But, if we consider fairly conventional architectures, we can see that very awkward instruction sets, like Intel's, can end up having higher performance than much cleaner instruction sets like Sparc and Alpha specifically because of the size of the implementation budget Intel has at their disposal.

(PS after grading the question) A general comment about your answer: if it's an obvious result of your response that the IA-32 architecture can't be made to perform well, in the face of the fact that implementations of that architecture have the highest performance available today, you're in trouble.

2. Consider the following assembly language code. For three-operand instructions, the result is the first operand.

```
add r0, r1, r2
sub r4, r1, r2
add r3, r0, r4
add r0, r4, r3
```

Yes, *r2* was a typo and I meant *r2*.

(a) Show all of the dependencies and antidependencies in this code.

Dependencies shown as arrows, antidependencies as dashed arrows

```
add r0, r1, r2
sub r4, r1, r2
add r3, r0, r4
add r0, r4, r3
```

Missing antidependency or dependency -2

(b) If you have an infinite number of decoders, functional units, etc. and

- i. Fetching an infinite number of instructions from memory takes one cycle
- ii. Decoding them all takes one cycle
- iii. Executing an instruction, and making its result available to another instruction requiring it, takes one cycle

Draw a Gantt (timing) diagram showing how many cycles the code will take to execute.

```
add r0, r1, r2 _____
sub r4, r1, r2 _____
add r3, r0, r4 _____
add r0, r4, r3 _____
```

A whole bunch of people decided to either decode these instructions into μ ops or add some load instructions, or both. Why? That wasn't in the question.

Missing part b -12

3. Traditionally, a computer's virtual memory architecture is fixed, and translation lookaside buffer (TLB) management is in hardware. So, for instance, Intel documents their two-level page table structure, and the presence of the TLB is all but invisible to the implementer of the operating system.

A recent development has been management of the TLB in software. In this model, there is no fixed page table structure; instead, any time a TLB miss occurs there is a software interrupt to update the TLB entries, and the structure of the page tables (whether they be multi-level forward page tables, inverted page tables, hashed page tables, or some other structure) is handled completely by the operating system.

Discuss the strengths and weaknesses of the two approaches.

The principle advantage is flexibility. We can use any sort of multilevel forward page table, or inverted page table, or other structure that might be appropriate to our OS design. If we have a particular application like a database, this can also give us the opportunity to have a custom structure that might be more efficient than any standard system, as it's customized for the application.

On the other hand, if every TLB miss traps to software, it's very unlikely to be as fast as a hardware (or partially hardware) solution. Superficially it might look dangerous, since if we move the TLB management code out of the TLB we won't be able to handle a TLB miss anymore, but this problem isn't really any worse than the existing problem that we have to be careful not to page our virtual memory management out to disk.

4. How many 4×4 switch boxes would be required to implement a 64×64 dynamic hypercube network?

This would require $64/4 = 16$ switch boxes per stage, and $\log_4 64 = 3$ stages, for a total of 48 boxes.

For some reason, quite a few people wanted to tell me how many switch boxes were required for a Benes network. I gave them 10 points.

5. Consider the following parallel processes P1, P2, and P3 executing on three processors:

P1	P2	P3
x = 1; y = 3; z = x - y;	y = 4; x = 2; z = x + y;	x = 3; y = 7; z = y - x;

(x, y, and z are all shared variables)

- (a) Add one or more synchronization operations S to all three processes which will guarantee that the final value of variable z is 5.

In solving this one, I arbitrarily decided to see if I could get P2 to give me the answer.

I noticed that P1 sets y to 3 and P2 sets x to 2, so if I could make sure P2 saw those values of x and y then P2 would give me the answer I was looking for. Inserting the first sync point in each process accomplished this; it made sure that those assignments happened after the sync and covered any prior assignments to the variables.

So now, I had to make sure P2's assignment to z happened last. Inserting the second sync point assured that. So my final answer looks like this.

P1	P2	P3
x = 1; S; y = 3; z = x - y; S;	y = 4; S; x = 2; S; z = x + y;	x = 3; y = 7; S; z = y - x; S;

- (b) Using the memory read/write/sync notation we've been using in this class (for instance, in the solution to HW3 question 3), show the operations which will be performed.

Of course, precisely what values are read depend on the memory model you're using. I used weak consistency, assuming a processor could see its own writes before the syncs communicate them to the others. Also, of course, if two processors both write to a single variable and then a sync happens as we see in the first epoch, which one "wins" is undefined - in fact, I don't think there is even a requirement that a single write needs to be the global winner in this case! Though I do have a single winner.

P1	w(x)1	S	w(y)3	r(x)1	r(y)3	w(z)-2	S			
P2	w(y)4	S	w(x)2				S	r(x)2	r(y)3	w(z)5
P3	w(x)3	w(y)7	S	r(y)4	r(x)1	w(z)3	S			

6. Error Correcting Codes

- (a) Suppose you have an eight bit byte using a SECDED Hamming correcting code, as follows:

M8	M7	M6	M5	C8	M4	M3	M2	C4	M1	C2	C1	P
0	1	0	1	0	1	0	1	1	1	0	1	1

This may have no errors, a one-bit error, or a two-bit error. Which is it? If it has a one-bit error, correct it.

Bit Addr	M8	M7	M6	M5	C8	M4	M3	M2	C4	M1	C2	C1	P	Syindr
Par	0	1	0	1	0	1	0	1	1	1	0	1	1	0
C8	0	1	0	1	0									0
C4	0					1	0	1	1					1
C2		1	0			1	1			1	1	0	0	0
C1		1		1		1		1		1		1		0

The syndrome is non-zero, but the parity is correct. Therefore, we have a two-bit error.

Correcting M3 -5

- (b) Suppose a value using a SECDED Hamming code has a three bit error. When the receiver tries to read this value:

- i. How many bits will it "think" are in error?

Since the Hamming distance between two correct codes in this case is 4, and we are a distance of 3 from one valid code, we are a distance of 1 from another. So it will "think" there is a one-bit error.

One student noticed that it might conclude there is an error in a nonexistent bit (bits 13-15 if we've got the eight bit scheme we've used as an example), from which it could deduce a three bit error. Nice!

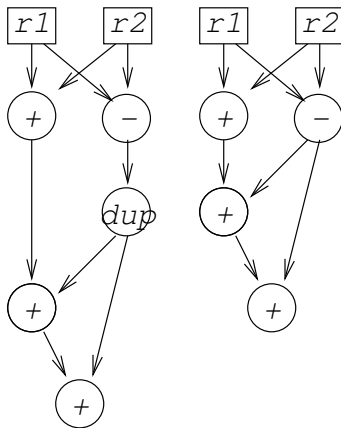
- ii. What will happen when it tries to correct it?

It will correct the bit it "thinks" is in error, and send something completely wrong to the receiver.

7. Consider the code from Question 2. Suppose you were to translate this code for execution in a dataflow computer.

In writing this question, I missed specifying something: what to do if a result was used by two downstream instructions? The best answer (because it corresponds to real dataflow machines) is to have a “dup” instruction that makes two copies of its input so they can be sent to two instructions. Another approach is to just define instructions as being able to have more than one output. I’ll give both answers.

(a) Draw a dataflow diagram for the code.



I didn't take points for people who put in register names for placeholders - unless they formed a cycle with r0.

(b) Assume the dataflow computer’s arithmetic instructions each have an address, and have the following fields:

- i. The operation to be performed (+ and - in this case)
- ii. The address of the instruction to which the result should be sent, and whether the result should be the instruction’s first or second operand.

What are the dataflow computer’s instructions? Assume the instruction addresses are 1, 2, 3, and 4.

Addr	Op	Res1	Op1	Res2	Op2
1	+	3	1		
2	-	3	2	4	2
3	+	4	1		
4	+				

Addr	Op	Res1	Op1	Res2	Op2
1	+	3	1		
2	-	5	1		
3	+	4	1		
4	+				
5	dup	3	2	4	2

Obviously, this part of the problem actually assumed the two-result version, since I specified the addresses of four instructions