

CS 473

Final Exam

Solutions

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 10:00 to finish the exam.

1. (20 points) Floating point:

- (a) Convert the human-readable decimal number 62.375 to IEEE floating point format.5

62 = 63-1, so in binary it's 111110. 0.375 in binary is .011, so we have 111110.011. Normalizing makes it 1.11110011×2^5 . The sign is 0, the exponent field is $127 + 5 = 132$, or 10000100 in binary. Putting it together, and remembering to leave off the phantom bit, gives 0 10000100 111100110-0 in binary or 0x42798000.

- 2 *Bit mysteriously appears without explanation*
- 4 *Converted .875 wrong, no work shown*
- 2 *Sign wrong*
- 2 *Didn't assemble answer*

- (b) Add it to the IEEE floating point number 0xc1ab0000. Your final result should be a 32 bit (8 digit) hexadecimal number.

Converting this to binary gives 11000001101010110-0; breaking it into fields becomes 1 10000011 01010110-0. Putting the phantom bit in makes the significand 1.0101011. The exponent is 1 less than the exponent from part 1a, so we need to right-shift the significand giving 0.10101011. It's negative so we subtract:

$$\begin{array}{r} 1.11110011 \\ -0.10101011 \\ \hline 1.01001 \end{array}$$

No renormalization is necessary, so we reassemble as 0 10000100 010010-0 in binary, or 42240000

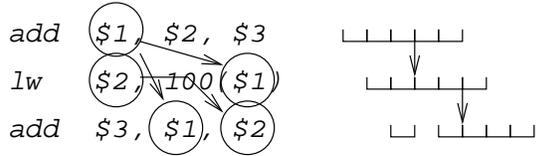
- 3 *Forgot to un-normalize*
- 2 *Got sign of smaller number*
- 2 *Added instead of subtracting*
- 1 *Forgot phantom bit*

2. (10 points) Consider the following sequence of MIPS instructions.

```
add $1, $2, $3
lw $2, 100($1)
add $3, $1, $2
```

(a) Draw arrows showing the dependencies in this code.

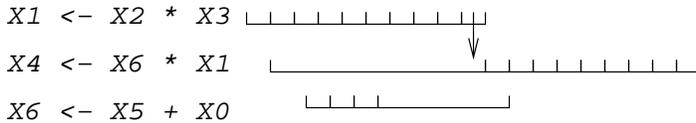
(b) Draw a Gantt (timing) chart showing how this code would be executed using the MIPS pipeline on page 492 of the text. Use arrows showing forwarding; be sure to show the forwarding in the correct pipeline stages.



- 1 Forward to wrong stage
- 1 Missing pipeline stage
- 2 Missed dependency
- 1 Extra dependency
- 2 Forgot to forward
- 1 Extra forward
- 2 No stall
- 2 Unnecessary stall
- 1 Instruction a cycle late or early

3. (10 points) Draw a Gantt (timing) chart to show how the following code would be executed on a CDC 6600. Use arrows to show forwarding.

```
X1 <- X2 * X3
X4 <- X6 * X1
X6 <- X5 + X0
```



- 2 Treat third instruction as 2nd order conflict
- 1 Wait too long on third instruction
- 1 Didn't show forwarding
- 3 Didn't treat third instruction as any conflict at all

4. (15 points) Here is some information from the memory of a computer using Intel's virtual memory scheme:

Address	Contents
25204b40	6366a404
2bd6a4d4	235c9800
4fdd349c	2bd6a043
57c47d60	2520407f
736445a8	57c47067
73644674	4fdd3007

Assume the PDDBR contains 73644000.

(a) What will happen if a process running in user mode attempts to write to virtual address 675274d4?

Directory entry is at 73644674 containing 4fdd3007; present, user-accessible, writeable.

Page table entry is at 4fdd349c containing 2bd6a043; present, writeable, but not user-accessible so we get a protection violation. If we hadn't had a protection violation, we would have written to 2bd6a4d4.

- 4 Thought protection fault happened at directory entry
- 2 thought "0" meant user-accessible
- 5 Stopped after one level
- 4 No sign they actually checked protections

(b) How is it possible for a page table entry to have a Dirty bit equal to 1, but an Accessed bit equal to 0?

The OS periodically resets accessed bits, so it can keep track of how recently a page was accessed. Once the dirty bit is set, it has to stay set.

- 5 If written to but not read from

5. (10 points)

(a) What is the largest that an 8-way set-associative cache can be and still let a system using Intel's virtual memory scheme do a simultaneous lookup in TLB and (physically addressed) cache?

The requirement for this to work is that the VM translation can't affect the index or offset bits, so we can only have a total of 12 index and offset bits for a total of 4K. If it's 8-way set-associative we can get an 8-fold increase, so the limit is 32K.

- 4 Wrong number no justification
- 2 2^{10} instead of 2^{12} , no idea why
- 1 $<$ instead of \leq (so 11 bits)
- 2 Calculated for direct-mapped

(b) Suppose the largest possible cache from part 5a uses a 256 byte block size. How is the address broken up into tag, index, and offset? Suppose an attempt is made to look up address 0x13a48df8 in this cache. What will the tag, index, and offset be for this address?

The 256 byte blocksize fixes the offset at 8 bits leaving 4 for the index and 20 for the tag. For the given address, this works out to

*Tag: 13a48
Index: d
Offset: f8*

- 2 didn't show breakdown

(c) That's a pretty big block size (in part 5b). What are the drawbacks to such a big block size?

The two major drawbacks would be a very expensive cache miss (due to the amount of memory to be read; it might be expensive in terms of bus width if the bus is actually 256 bytes wide, or more likely expensive in time to transfer all that data through a narrower bus), and since we only end up with 16 sets a high likelihood that blocks would be brought in but then removed before all the data in the block was accessed.

- 2 Slower to access data in block

6. (15 points) Calculate the parity and check bits for the eight-bit binary number 01100100. Express your result as a 13 bit binary number with the bits in the order M8 M7 M6 M5 C8 M4 M3 M2 C4 M1 C2 C1 P.

I'm presenting the bits I'm using to calculate the check and parity bits in the same order as in the problem:

$$\begin{aligned} C8 &= 0^1 1^1 0^1 = && 0 \\ C4 &= 0^0 0^1 1^0 = && 1 \\ C2 &= 1^1 1^0 0^1 1^0 = && 1 \\ C1 &= 1^0 0^0 0^0 0^0 = && 1 \\ P &= 0^1 1^1 1^0 0^0 0^0 1^0 1^0 1^0 1^1 = && 0 \end{aligned}$$

So we get 0110001010110

- 3 No idea where he got the numbers he used to compute C, P
- 3 Not enough numbers used in calculating C, P
- 5 Concluded an error???
- 1 Didn't include check bits in parity

As I was grading these, I finally figured out that quite a few people thought "8 bit binary number" meant some 8 bits (most frequently the first 8 or the last 8) taken from a 13-bit number that also included the check and parity bits, and all the bits that weren't included were 0. Then they concluded there was an error... no, I just asked you to compute the check and parity bits for a particular 8 bit data word.

7. (20 points) A particular computer system, and its associated disk drive, has the following characteristics:

Processor speed:	2 billion instructions/sec
Average disk seek time:	10 msec
One-track seek time:	1 msec
Rotation speed:	6000 RPM
Data transfer speed:	100,000,000 bytes/sec

This computer system is executing a program that requires it to repeatedly execute transactions requiring the following sequence of operations:

- Spend 100,000 instructions calculating its next sequence of reads and writes.
- Read 5 blocks from the disk. The first block is at a random location on the disk; the remaining 4 are all on adjacent tracks. Amazingly, they are offset from one another just right so the only time required to get from one block to the next is the track-to-track seek time. Each block is 10,000 bytes.
- Spend 10 million instructions processing the data.
- Write one (10,000 byte) block to a random location on disk.

- (a) How long does it take to perform one of these transactions?

Computation 1: $1 \times 10^5 / 2 \times 10^9 = 5 \times 10^{-5}$

Disk read: $10 \times 10^{-3} + (1/2) \times (60/6000) + 4 \times 10^{-3} + 5 \times 10^4 / 1 \times 10^8$
 $= 10^{-2} + 5 \times 10^{-3} + 4 \times 10^{-3} + 5 \times 10^{-4} = 19.5 \times 10^{-3}$

Computation 2: $1 \times 10^7 / 2 \times 10^9 = 5 \times 10^{-3}$

Disk write: $10 \times 10^{-3} + (1/2) \times (60/6000) + 1 \times 10^4 / 1 \times 10^8 = 15.1 \times 10^{-3}$

Total is $5 \times 10^{-5} + 19.5 \times 10^{-3} + 5 \times 10^{-3} + 15.1 \times 10^{-3} = 39.65 \times 10^{-3} \text{seconds}$

I'm pretty much grading part 7a based on a successful setup. Every missing term cost 2 points.

- (b) You are given the option of doubling the performance of one of the following aspects of the system: the processor speed, the speed the head can move between tracks (so the seek time would be halved), or the disk transfer rate. Which one should you pick? Why, in the practical world, is that the hardest one of the three to improve?

Looking at the components of the answer up above, processing takes 5.05 msec, data transfer takes .6 msec, and seeks take 24 msec (rotational latency takes the last 10 msec, but I didn't give that as an option). Clearly, seeks are the biggest component of the time, so speeding them up would have the greatest impact. They are the hardest, since they're a mechanical system.

Several people suggested the disk RPM should be sped up. I decided to accept this as a possible answer even though it wasn't one of the three options I gave... all the same, it only cost 10 msec, so it wasn't the best choice.

- 2 "Hardware," not specifically mechanical
- 5 Transfer rate
- 2 RPM
- 5 CPU is a mechanical system (!)