CS 473
Midterm Exam
February 28, 2003

The following exam is open book and open notes. You may feel free to use whatever additional reference material you wish, but **no calculators** 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 you arrived at your answers
- be succinct. You may lose points for facts that, while true, are not relevant to the question at hand

You have until 10:20 to finish the exam.

1. *(30 points)* Multiply the two following IEEE floating point numbers together: `41300000` $\times$ `40500000`. Express your result as an eight-digit hexadecimal number. Convert it to human-readable decimal format.

   `41300000`, or

   `0100 0001 0011 0000 0000 0000 0000 0000`, or

   `0 10000010 01100000000000000000000`

   Sign: 0

   Exponent: `10000010`

   Significand: `1.011`

   `40500000`, or

   `0100 0000 0101 0000 0000 0000 0000 0000`, or

   `0 10000000 10100000000000000000000`

   Sign: `0`

   Exponent: `10000000`

   Significand: `1.101`

   Result sign: `0`

   Exponent: `10000010 + 10000000 - 01111111 = 10000011`

   Significand:

   ```
    1.101
   ×1.011
        1101
       1101
      0000
    1.101
   10.001111
   ```

   Renormalizing, exponent = `10000100` and significand = `1.0001111`. Combining, we have

   `0 10000100 0001111`, or

   `0100 0010 0000 1111 0000 0000 0000 0000`

   `420f0000`

   Translating to decimal, `1.0001111` $\times 2^5$ = `10001.11` = 35.75.

2. *(10 points)* Translate the following MIPS instructions to machine code. Expressyour answer as an eight digit hexadecimal number.

   (a) `xor $1, $2, $3`

   an R-format instruction; fields are `0, 2, 3, 1, 0, 38`$_{10}$. so we get

   `000000 00010 00011 00001 00000 100110`, or

   `0000 0000 0100 0011 0000 1000 0010 0110`, or

   `00430826`

(b) `beq $7, $9, 0x300`

an I-format instruction; fields are

`4, 7, 9, 00c0` (remember division by 4), or

`000100 00111 01001 0000 0000 1100 0000`, or

`0001 0000 1110 1001 0000 0000 1100 0000`, or

`10e900c0`

(c) `lw $t0, 200($s1)`

A small oops on this one: I should have specified the offset as hexadecimal; when I was asked about it during the exam, it was too late to try to fix it. So I'll accept either answers that treat 200 as hexadecimal, or as decimal (c8 hexadecimal). Another oops, and something that near as I can tell nobody has spotted before this semester, is that there is a typo in the book: the inside back cover has the opcode for `lw` as 23_{16}. while page A-54 shows 23_{16} as `lwl`, with lw as 24_{16}. As if that weren't enough, A-66 shows both `lw` and `lwr` as 23_{16}, and 24_{16} as `lbu`! So I'll take either 23_{16} or 24_{16}.

Another I-format instruction, fields are

24_{16}, `17, 8, c8` (no division this time), or

`100100 10001 01000 0000 0000 1100 1000`, or

`1001 0010 0010 1000 0000 0000 1100 0000`, or

`922800c8`

If you treated the 200 as hexadcimal, the final answer is

`92280200`

If you used an op code of 23, the corresponding final answers are `8e2800c8` and `8e280200`.

3. *(10 points)* Translate the following machine code instructions to MIPS assembler. You can just use raw register numbers (like `$7`), you don't need to translate to the MIPS assembler conventions (like `$a3`). You may express constants in either decimal or hexadecimal, but please use C syntax (`0x`) to tell me if it's hexadecimal.

   (a) `0x02a72824`

   `000000 10101 00111 00101 00000 100100`

   op code `0` so it's R format (and I actually split up the rest of the fields shown above after I'd worked that out); func is 36_{10}, *so the instruction is*

   `add $5, $21, $7`

   Quite a few people translated this according to the usage in Table A.10. That wasn't necessary, but of course I accepted it. Given how many people had a hard time finishing the exam, it seems like the time spent looking this up could have better been spent elsewhere, though:

   `add $a1, $s5, $a3`

   (b) `0x2d234321`

   `001011 01001 00011 0100 0011 0010 0001`

   op code `b`, so it's an

   `sltiu $3, $9, 0x4321`

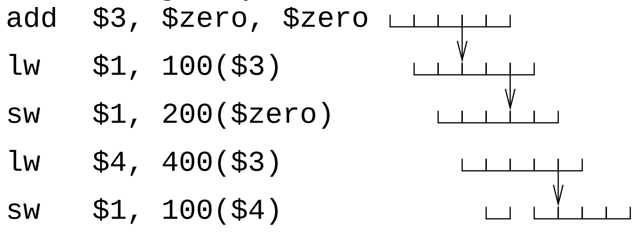
   Again, translated, this is

   `sltiu $v1, $t1, 0x4321`

4. *(20 points)* Consider the following MIPS code fragment:

```
add  $3, $zero, $zero
lw   $1, 100($3)
sw   $1, 200($zero)
lw   $4, 400($3)
sw   $1, 100($4)
```

   (a) Draw a Gantt (timing) chart showing the execution of this code, assuming all possible forwarding and assuming loads stall instead of using a delayed load. Use arrows to show forwarding between the instructions.



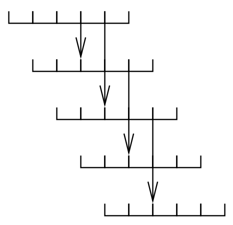
```
add  $3, $zero, $zero
lw   $1, 100($3)
sw   $1, 200($zero)
lw   $4, 400($3)
sw   $1, 100($4)
```

We can forward `$3` to the first `lw`, for use in address generation. The forwarding from the first `lw` to the first `sw` may be a surprise; we can do it because the register is being stored rather than being used for arithmetic. We do have to stall the final `sw`, since in this case the value read in the previous instruction is used in the address calculation.

(b) Reorder the code so that it can run as fast as possible.

All we have to do is separate the last `lw` from its `sw`, while making sure we don't move the `lw` so far forward that it's before the `add`. Just moving it ahead of the preceding `sw` will do it, like this:

```
add  $3, $zero, $zero
lw   $1, 100($3)
lw   $4, 400($3)
sw   $1, 200($zero)
sw   $1, 100($4)
```



This saves one cycle.

5. *(30 points)* The book has a possible superscalar version of the MIPS on page 512. This superscalar machine has two pipes: one for memory instructions, and tee other for everything else. Suppose the following two instructions occur in a program:

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

Assuming all possible forwarding, would it be possible to issue these instructions simultaneously? If not, why not? If so, modify the following copy of Figure 6.58 to show the forwarding that would be necessary.

Yes. The key observation is that the `add` instruction generates a new value for `$1` during the execute cycle, and the `sw` instruction uses it on the memory cycle. So it can be forwarded from the top ALU output line to the memory's Write Data line.

The most consistent mistake was probably people thinking `$1` had to be sent to the address generation ALU, instead of the write data line. If that were what was needed it would indeed not be possible (and people who made that mistake were about evenly divided as to whether they thought it would be possible but the pipeline slower and people who thought it would not be possible).