CSE30 — Midterm

Yee **Fall '97**

Name / Login: Answer sheet

There are a total of 17 questions on 13 pages. There are 100 points possible. It is unlikely that you will finish the entire exam. Wait until the instructor has passed out exams to everybody before you start. Advice: skim through the entire test to determine which of the problems you can solve quickly and work on those first, rather than getting stuck on a hard problem early and wasting too much of your time on it.

When you can start, you should first make sure that you have all the pages, and write your name and your login name at the top of first page, and your login name on the top of *all subsequent pages*. Pages of this exam will be separated and graded separately — if you fail to write your name at the top of a page, you will not receive credit for answers on that page. **Write clearly**: if we cannot read your handwriting or your pencil smudges, you will not properly get credit for your answers.

This exam is open book, open notes (1 sheet of paper), but *not* open people (no scalpels, please). You may look at your *own* books and notes all you want. You may **not** look at anybody else's books, notes, exam, or otherwise obtain help from another human being, artificial intelligence, or space alien. If we see your eyeballs wandering, you will get a zero for the exam. If you must look away from your exam/notes to think, look up into space.

No electronic computation aids are allowed.

1. **[Base Conversion]** Given a string of digits $d_{k-1}d_{k-2}\ldots d_2d_1d_0$ in a certain base b, where $0 \le d_j < b$ for $j = 0, \ldots, k-1$, written as $d_{k-1}d_{k-2}\ldots d_2d_1d_0\,_{(b)}$, what is the corresponding integer? Write this in a mathematical notation. Also write down this number multiplied by b as a string of digits.
(4pt)

Ans:

The number is

$$\sum_{i=0}^{k-1} d_i b^i$$

b times this number is just $d_{k-1}d_{k-2}\ldots d_2d_1d_0 0\,_{(b)}$.

2. [**Base Conversion**] Perform the following base conversions. For bases larger than $16_{(10)}$, the individual digits are written as parenthesized base 10 numbers, e.g., $(17)(30)_{(72)} = 17_{(10)} \times 72_{(10)} + 30_{(10)}$.

1: $\texttt{FACE0FF}_{(16)} = ?_{(2)}$

2: $\texttt{2174563582}_{(9)} = ?_{(3)}$

3: $(\texttt{74})(\texttt{34})(\texttt{3})_{(81)} = ?_{(3)}$

4: $\texttt{12211022}_{(3)} = ?_{(27)}$

5: $\texttt{33653337357}_{(8)} = ?_{(16)}$

(15pt, 3 each)

Ans:

1: $\texttt{FACE0FF}_{(16)} = \texttt{1111 1010 1100 1110 0000 1111 1111}_{(2)}$

2: $\texttt{2174563582}_{(9)} = \texttt{02 01 21 11 12 20 10 12 22 02}_{(3)}$

3: $(\texttt{74})(\texttt{34})(\texttt{3})_{(81)} = \texttt{2202 1021 0010}_{(3)}$

4: $\texttt{12211022}_{(3)} = (\texttt{5})(\texttt{22})(\texttt{8})_{(27)}$

5:

$$\begin{aligned} \texttt{33653337357}_{(8)} &= \texttt{011 011 110 101 011 011 011 111 011 101 111}_{(2)} \\ &= \texttt{0 1101 1110 1010 1101 1011 1110 1110 1111}_{(2)} \\ &= \texttt{DEADBEEF}_{(16)} \end{aligned}$$

3. [**Micro-architecture**] What is an ALU? Explain what it does.
(4pt)

Ans:

An ALU is the Arithmetic and Logic Unit. It is the circuits that perform operations such as add, subtract, multiply (sometimes there is a separate multiply unit), and various bit-wise operations.

4. [**Micro-architecture**] What is the purpose of a cache? Explain what it is and how it achieves its purpose.
(5pt)

Ans:

A cache improves overall performance of the computer by transparently making memory accesses faster most of the time. It is fast memory that is located closer to the processor, and contains copies of portions of main memory contents. When the processor attempts to access cached memory, a *cache hit* occurs and the access is fast, since the memory access does not have to travel to the DRAM over the system bus; when the processor attempts to access memory that has not been cached, a *cache miss* occurs and the cache forwards the access to the DRAM, saving (caching) a copy of the result for subsequent use.

If p is the probability of a cache hit, then the expected memory access time is $p \cdot t_{\text{hit}} + (1-p) \cdot t_{\text{miss}}$; typically caches are designed to have a very high p (depends on size and program mix), e.g., 0.99, and $t_{\text{hit}} << t_{\text{miss}}$, so including caches greatly improves overall performance of computers.

5. **[Number representation]** Compute the two's complement of the following 16-bit numbers:

1: `0x3248`

2: `0x9321`

Negate the following numbers stored in 16-bit registers:

3: `0x328`

4: `0xA34C`

In all 4 cases, mark which results would be interpreted as a negative number when interpreted as a 16-bit two's complement number.
(8pt, 2 each)

Ans:

Taking the two's complement of a number is the same as negating it.

1: `0x3248` $\rightarrow$ `0xCDB8` (negative)

2: `0x9321` $\rightarrow$ `0x6CDF`

3: `0x328` $\rightarrow$ `0xFCD8` (negative)

4: `0xA34C` $\rightarrow$ `0x5CB4`

6. [**Number representation**] Suppose you have a 32-bit number in a register, and its hexidecimal representation is 0x80000000. Is this number positive or negative when viewed as a two's complement number? What happens when you negate it? Is the result of the negation positive or negative when viewed as a two's complement number?
(7pt)

Ans:

The number is negative when viewed as a two's complement number, since the high-order bit is set. The result from negating it is also 0x80000000. An overflow occurred during the negation, because the result can not be represented as a 32-bit two's complement number (it's too big). The result would be interpreted as the same as the original negative number.

---

7. [**One Instruction Computer**] Define the `subz` instruction.
(4pt)

Ans:

```
subz a,b,c
```

is equivalent to the following C-like pseudo-code:

```
mem[a] = mem[a] - mem[b];
if (mem[a] == 0) pc = c;
else pc = pc + 1;
```

8. [**One Instruction Computer**] Convert the following `subz` program fragment to its hexidecimal representation.

```
    subz t,t,next
    subz t,a,next
    subz t,b,next
    subz c,c,next
    subz c,t,next
    subz t,t,this
a:  0x123456789a
b:  0x23456789abcd
c:  0
t:  0
```

Assume that the first instruction will be in memory location 0. You do not need to add comments to this code, but you should be explicit about how you assigned addresses and how you arrived at your result.
(8pt)

Ans:

```
       9       9       1  ; 0      subz t,t,next
       9       6       2  ; 1      subz t,a,next
       9       7       3  ; 2      subz t,b,next
       8       8       4  ; 3      subz c,c,next
       8       9       5  ; 4      subz c,t,next
       9       9       5  ; 5      subz t,t,this
    0x12  0x3456  0x789a  ; 6  a:  0x123456789a
  0x2345  0x6789  0xabcd  ; 7  b:  0x23456789abcd
       0       0       0  ; 8  c:  0
       0       0       0  ; 9  t:  0
```

9. [**Endianism**] In a little-endian machine, if we loaded the first four bytes of a string into a register as an integer, what would be the value in that register? Let the string be "ABCD". You may assume that the string is word aligned. The ASCII value of the letter are: "A" = 0x41; "B" = 0x42, "C" = 0x43, "D" = 0x44. Express the register contents as a number.
(4pt)

Ans:

A little-endian machine would store the low order byte of an integer into the lowest memory address; so a load would load the byte with the lowest address as the lower order byte of the integer. C strings are character arrays, so the "A" will be the character with the lowest address. Thus, the result will be 0x44434241.

---

10. [**MIPS**] What are the MIPS `t` and `s` registers used for? In what way are they different from each other?
(5pt)

Ans:

Both the `t` and `s` registers are for temporaries. The `t` registers are *caller-saved* registers, since by convention a subroutine is allowed to use them. The `s` registers are *callee-saved* registers; a routine can call a subroutine and expect that these registers' contents will be preserved when the subroutine returns.

---

11. [**RISC and CISC**] Give an example of a processor with a RISC architecture and an example of processor with a CISC architecture.
(3pt)

Ans:

The MIPS architecture is a RISC, and the R2000 is an implementation of that architecture; a 486 is a processor that implements the x86 (or IA-32) architecture, which is a CISC architecture.

12. [**Computer Architecture / OS**] What are page faults? Why do their existence influence the design of a processor?
(6pt)

Ans:

A page fault occurs when there is no virtual-to-physical address translation possible because the memory contents were paged out to disk. When a page fault occurs, the operating system must find some free memory, read the page back in, fix the virtual-to-physical address mapping to translate the virtual address to the (new) physical address, and restart the instruction that caused the page fault.

Because page fault handling requires that the processor save enough state in order to restart instructions, this complicates the processor design. A processor with complex addressing modes and complicated instructions (such as the x86's `REP` prefix or the Vax's polynomial evaluation instruction) require a lot of state to be saved/restored. RISC designs avoid complexity that are rarely useful, so tend to be load/store architectures, where only explicit `load` and `store` instructions are used to reference memory, and all other instructions only involve registers.

13. [**Translating C to assembly**] Translate the given the following C code into an equivalent series of MIPS assembly language instructions. You may assume that the C variables are in the correspondingly named registers. Indicate where the code that preceeds the loop, the code that comprise the body of the loop, and the code that follows the loop would be located in your equivalent MIPS code. Efficiency matters.

```
code that precedes loop
for (t0 = 0; t0 < t1 && t2; t0++) {
        loop body
}
code that follows loop
```

(9pt)

Ans:

```
        code that precedes loop
        li $t0,0
        b test
loop:   loop body
        addi $t0,$t0,1
test:   bge $t0,$t1,done
        bne $t2,$zero,loop
done:   code that follows loop
```

14. [**Stack Frames**] Write the MIPS assembly language equivalent for the following function:

```
int fib(int n)
{
    if (n <= 1) return 1;
    else return fib(n-1) + fib(n-2);
}
```

(9pt)

Ans:

```
fib:        sub $sp, $sp, 16
            sw $fp, 4($sp)
            add $fp, $sp, 16
            sw $ra, -8($fp)
            bgt $a0, 1, rec_fib
            li $v0, 1
            b fib_done
rec_fib:    sw $a0, 0($fp)
            sub $a0, $a0, 1
            jal fib
            sw $v0, -4($fp)
            lw $a0, 0($fp)
            sub $a0, $a0, 2
            jal fib
            lw $a0, -4($fp)
            add $v0, $v0, $a0
fib_done:   lw $ra, -8($fp)
            lw $fp, 4($sp)
            add $sp, $sp, 16
```

15. **[Proofs of Correctness]** The loop invariant for the fibonacci function is `f0 = fib(i)` and `f1 = fib(i+1)`. This invariant holds at just before the test `i < n` (marked with a bullet [•]). Prove that this is indeed a loop invariant, and use it to prove that the function is the fibonacci function. (The definition of the fibonacci function is `fib(0) = fib(1) = 1`, and `fib(n+2) = fib(n+1) + fib(n)`.)

```
fib(unsigned int n)
{
      unsigned int i, f0 = 1, f1 = 1, t;
      for (i = 0; • i < n; i++) {
            t = f0 + f1;
            f0 = f1;
            f1 = t;
      }
      return f0;
}
```

(9pt)

Ans:

Base case: when `i = 0`, `f0 = 1`, `f1 = 1`, and so the invariant holds. Next, we assume that the invariant holds when $\mathtt{i} = k$, so $\mathtt{f0} = \mathtt{fib}(k)$ and $\mathtt{f1} = \mathtt{fib}(k+1)$. We trace the execution of the code once through the loop body, and see that

$$\begin{aligned} \mathtt{t} &\leftarrow \mathtt{f0} + \mathtt{f1} \\ &= \mathtt{fib}(k) + \mathtt{fib}(k+1) \\ &= \mathtt{fib}(k+2) \\ \mathtt{f0} &\leftarrow \mathtt{f1} \\ &= \mathtt{fib}(k+1) \\ \mathtt{f1} &\leftarrow \mathtt{t} \\ &= \mathtt{fib}(k+2) \end{aligned}$$

and then `i` is incremented as part of the loop control, getting `k+1`. At the next encounter with •, we have $\mathtt{i} = k+1$, $\mathtt{f0} = \mathtt{fib}(k+1)$, and $\mathtt{f1} = \mathtt{fib}(k+2)$, which is what we wanted and the invariant holds.

When the loop terminates due to the test failing, $i == n$ holds, so we have $\mathtt{f0} = \mathtt{fib}(n)$, which is what the code returns. Thus, the code correctly computes $\mathtt{fib}(n)$.

16. [**Extra credit (?)**] If you could choose the flavor of a cream pie (vanilla, chocolate, or banana) to be thrown at the professor, which would you choose? Why?
(0pt)

Ans:

17. [**For Your Amusement After The Exam**] To really efficiently calculate the `fib` function, we use a little more mathematics. We first express the `fib` function as matrix operations:

$$\texttt{fib}(n) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

where $\cdot$ is vector dot product, and exponentiate via the fast exponentiation algorithm, using the binary expansion of the exponent and repeated squarings. (If you know what eigenvalues are, then you can also convert to a diagonal matrix of eigenvalues.)
(0pt)

Ans:

(Use the back of this page for overflow. Clearly mark which problem's answer is being overflowed on both this page and the page containing the original question.)