

EECS 370
Exam 1
Fall 2003
October 9, 2003

Name:

University of Michigan username:
(NOT your student ID number!)

Honor code pledge: I have neither given nor received aid on this examination, nor have I concealed any violations of the Honor Code.

Signature:

(examinations without a signed honor pledge will not be graded)

12 pages, 10 questions, 100 points

1. True/False (12 points):

Circle TRUE or FALSE for each statement.

- (a) TRUE FALSE It is theoretically possible to implement any boolean function in only two levels of digital logic.
- (b) TRUE FALSE In a load/store architecture, only loads and stores are allowed to access data memory.
- (c) TRUE FALSE A machine cycle is a process through which one machine instruction is executed.
- (d) TRUE FALSE All functions must save callee save registers when they are called, and load their values back before returning.
- (e) TRUE FALSE A processor without floating point instructions cannot perform floating point computations.
- (f) TRUE FALSE Optimizing a single cycle datapath for the average case will improve performance.

2. Instruction formats (10 points):

Suppose you revised the LC-2K3 instruction set architecture such there are 200 different opcode encodings and 40 registers but instructions are still 32 bits. In order to accommodate this change, the format for I-type instructions has changed to incorporate the new opcodes and registers. All of the remaining bits are used for the signed offset field - there are no longer any unused bits.

If you have a `beq` instruction at address 10000 (base ten), what is the range of target addresses you can jump to under this new format?

3. Floating point multiplication (*10 points*):

Consider multiplication on a processor that uses IEEE 754 standard single precision (32 bit) floating point arithmetic. A floating point register on this processor contains the hexadecimal value 0xbec00000. This value is to be multiplied by 36.0 (decimal).

(a) What is the floating point representation of 36.0 decimal? (You may state your answer in either hexadecimal or binary.)

(b) The multiplication is performed, leaving the product in the floating point register. What is the correct representation of this product? (Again, your choice of either hexadecimal or binary.)

(c) What is this product value in decimal?

4. MIPS pseudo-instructions (*10 points*):

Suppose you are creating a new MIPS pseudo-instruction called **lwmi**. Its functionality is similar to that of a load word (**lw**) instruction, but **lwmi** uses memory indirect addressing. **lwmi** takes two arguments: a destination register and a label that represents the address. For example:

```
lwmi $t1 umich
```

is equivalent to:

```
$t1 = Memory[Memory[umich]]
```

Using real MIPS instructions (no pseudo-instructions), write the assembly language expansion for:

```
lwmi $t2 goblue
```

where goblue is at address 0x34A28143.

In MIPS, register **\$at** is used for temporary values during the expansion of psuedo-instructions and you can't assume that **\$at** is initially zero. Specify immediate values in hexadecimal. For full credit, you must use as few instructions as possible.

5. Adder design (10 points):

A 3-bit adder has seven inputs: A_2, A_1, A_0 (input bits for operand A), B_2, B_1, B_0 (input bits for operand B), and C_0 (carry-in). The adder has five outputs: S_2, S_1, S_0 (sum bits), P (propagate bit), G (generate bit). The propagate and generate bits can be used to create a carry lookahead adder such that C_3 can be determined using $P, G,$ and C_0 .

(a) Give an equation for determining C_3 in terms of $P, G,$ and C_0 .

(b) Give an equation for how the 3-bit adder determines P in terms of its seven inputs.

(c) Give an equation for how the 3-bit adder determines G in terms of its seven inputs.

6. Object files and linking (10 points):

In the following MIPS program, determine the instructions that will need relocation entries in the resulting object file. All registers adhere to the MIPS register convention. The labels foo and bar refer to the starting address of functions foo and bar, respectively.

```

        lw      $t3, 12($gp)      # instruction 1
        lw      $t5, 4($sp)      # instruction 2
L1:     addi    $t5, $t5, 3       # instruction 3
        addi    $t3, $t3, 1      # instruction 4
        beq    $t3, $t5, L1      # instruction 5
        jal    foo               # instruction 6
        add    $t4, $t3, $t5     # instruction 7
        la     $t6, bar          # instruction 8
        jr     $t6               # instruction 9
        sw     $v0, 0($gp)       # instruction 10
        sw     $v1, 4($sp)       # instruction 11
        beq    $v0, $v1, L2      # instruction 12
        j      L1                # instruction 13
L2:     jr     $ra               # instruction 14
```

Write the instructions (using the numbers on the right) that need relocation entries:

7. Performance (10 points):

Determine the running time for the following LC-2K3 program. The clock period is 5 ns. The number of cycles that each instruction takes is:

add, nand	3
beq	4
lw, sw	8
jalr	6
noop, halt	2

```

        lw    0 1 one
        lw    0 4 four
loop    beq   3 4 next
        sw    3 3 array
        add   3 1 3
        beq   0 0 loop
next    beq   0 3 done
        noop
        lw    1 6 array
        beq   6 1 done
        lw    0 3 dAddr
        jalr  3 5
done    halt
dAddr  .fill done
one     .fill 1
four   .fill 4
array  .fill 0
```

8. Logic design (8 points):

Implement a 3-input majority gate. A 3-input majority gate is a logic component that takes three inputs (A, B, and C), and returns output Z as a logic 0 if the majority of the inputs (two or more) are logic 0, and returns a logic 1 if the majority of inputs are logic 1. Give your answer as a gate-level circuit with the input(s) and output(s) labeled. You may use only NOT gates, 2 or 3 input AND gates, and 2 or 3 input OR gates in your implementation. Your implementation should be efficient; keep the number of levels of logic to a minimum and avoid redundant or unused logic.

