CS 366 - Computer Architecture II
Midterm Exam #2 - Prof. Reed
Fall 2003

What is your name?: [Key] (0 points)

There are two sections:
   I. Short Questions ........ 40 points; (20 questions, 2 points each)
   II. Short Sections of Code ... 60 points; (10 questions, 6 points each)

----------
100 points total

This test is worth 20% of your final grade. This test is open book and open notes. You have 50 minutes.

I. True/False: (2 pts. each)

T  F  1. According to our textbook, MIPS stands for Millions of Instructions Per Second.
T  F  2. In addition to the 32 MIPS register file registers having mnemonic names such as $t3, the
    registers can be referred to by their number, starting from $1 up to and including $32.
T  F  3. In a recursive program in most cases (though not in every case) $ra needs to be stored on the
    stack.
T  F  4. A carry-out at the most significant bit after an addition of two signed numbers always indica-
    tes overflow.
T  F  5. IEEE 754 single-precision floating point numbers always have an implicit 1 corresponding to
    the 1 that is stored to the left of the decimal place for a binary floating point number.
T  F  6. Both overflow and underflow are possible with floating point numbers.
T  F  7. IEEE 754 single-precision floating point numbers are evenly spaced on the number line.
T  F  8. As part of the process to add floating point numbers it is necessary to realign the smallest of
    the two (so it is no longer in normalized form) in order to make the exponents the same.
T  F  9. In the IEEE 754 single precision floating-point format, just like in mathematics, any number
    to the 0th power is 1.
T  F  10. In the solution for the hardwired binary multiplier we discussed in class the shift register Q
    has double-duty, being used for two purposes.
T  F  11. In the solution for the hardwired binary multiplier we discussed in class a carry-out from an
    intermediate result gets lost, as it is not stored directly into the A shift register.
T  F  12. Having a CDR (Control Data Register) is a convenience, however it does not have any effect
    on the speed of the control unit since each instruction must run to completion before the next
    instruction is run.
T  F  13. The maximum number of opcodes that we can directly implement in Mythsim is 64.
T  F  14. There is at least one MIPS assembler instruction that can directly store a result of an ALU
    calculation into memory.
T  F  15. If necessary, the $at register could be used by a SPIM programmer, though certain instruc-
    tions would have to be avoided.
16. The value part (in this case 0xf1f1) of the instruction `li $t1, 0xf1f1` can actually be a 32-bit value. _true because it is implemented as a macro._

17. If we find in a microprogram that we are running out of registers, one approach we can take is to temporarily store some register values on the stack. _exactly one_

18. Each state in an ASM diagram corresponds to one or more microprogram statements.

19. Once an IR is implemented in a control unit design, the MDR is only used to write to memory, and all reading from memory is through the IR. _Data still comes in through the MDR simultaneously._

20. A microprogram instruction could write to all registers in a single instruction. _√_

---

**III. Short Answer:** (6 points each)
Where there are multiple steps, please circle your final answer. You must show your work for credit.

1. Given the following diagram of a latch:

   ![Diagram of a latch]

   what does `d` have to be and what does `enable` have to be in order to store a 0 in `Q`?

   ![Logical operator symbols]

   4. pick: `d: 0`

   5. pick: `enable: 1`

2. Consider the following diagram:

   ![Diagram of a logic circuit]

   Is this ever capable of ever giving an output of 1? Explain why or why not.

   Yes. When the clock goes from 0 to 1 there is a propagation delay of the signal going through the NOT gate, so when the clock is finally a 1 both inputs are 1 (and the output is 1) for a period of time equal to the propagation delay.
3. Is it necessary to have two addresses in a microprogram control word? Explain why or why not.

No. If we had only one address, we could implement:

```
if cond, goto address endif
```

but we would not be able to implement:

```
if cond, goto address else goto otherAddress endif
```

To get the behavior of a two-address microprogram control word we would have to use two single-address control words.

4. What operation does the following microcode implement? Write it as a register-transfer operation (e.g. \( r7 \leftarrow r3 \)).

\[
\begin{align*}
0 & \leftarrow r2 + r4 \\
r0 & \leftarrow r2 + r4 \\
r1 & \leftarrow r3 + r5 + c_{in} \\
r1 & \leftarrow r3 + r5
\end{align*}
\]

```
first we do this & check for carry-out
```

```
second we do this, appropriately using carry-out from first part to set carry-in, chaining these together.
```

5. Why is it harder in microcode to implement

\[
\begin{align*}
\text{STORE} & \ (r1) \ , \ r3; \quad \text{Mem}[r1] \leftarrow \ r3 \\
\text{LOAD} & \ r1, \ (r3); \quad \ r1 \leftarrow \ \text{Mem}[r3]
\end{align*}
\]

than it is to implement \( r_j \leftarrow \ r_k \)?

Be specific in your answer.

For \text{LOAD}, the destination is specified in \( r_j \), and the source is in \( r_j \). We can select \( r_j \), putting it on the A bus and easily passing it through the ALU using ADDA without \( c_{in} \) set.

For \text{STORE} we need to pass both \( r_j \) (on the A bus) and \( r_k \) (on the B bus) through the ALU. It is straightforward to pass \( r_j \) through as described above, however to pass \( r_k \) through the ALU we must OR it with a register that we've set to 0, so it is more involved, taking more steps.
6. Consider the Mythsim datapath. Why is `a_sel` 3 bits, while `ir_rj` only 2 bits? (These are both inputs input the virtual `a_sel`).

   `a_sel` is used internally in microcode instructions and needs to be able to reference all 8 registers, so needs 3 bits (\(2^3 = 8\)).

   `ir_rj` specifies a register number coming from an assembler instruction, where valid registers are only r0-r3. These 4 registers can be specified in only 2 bits (\(2^2 = 4\)).

7. From a hardware point of view, why is the exponent stored in excess-127 notation for IEEE 754 floating point numbers?

   So that the same hardware can be used for comparisons, used for signed numbers.

8. Consider the following value, using the IEEE 754 single precision floating-point format. What is the equivalent value as a **decimal** number?

   \[ \begin{array}{c}
   \text{0100 0010 1010 1100 0000 0000 0000} \\
   \end{array} \]

   - sign bit is positive
   - \(x + 127 = 133\)
   - \(x = 6\)
   - \(1.01011 \times 2^6\)
   - \(= 1010110\)
   - \(= 64 + 16 + 6\)
   - \(= 86_{10}\)
9. Without changing anything in the hardware (i.e. datapath or control unit or memory interface), propose a way to extend the number of available opcodes in Mythsim, as compared to the current limit on maximum number of possible opcodes. First give an intuitive explanation, then write your solution in pseudocode, not microcode (however each step must be readily implementable in microcode).

Intuition: One special opcode would be the EXTEND opcode. Its microcode would take the \texttt{CONST4} (or \texttt{CONSTS}) value from the instruction and use its value as a constant to check in a secondary jump table with the additional opcodes.

Pseudocode: 

10. Without changing anything in the hardware (i.e. datapath or control unit or memory interface), propose a way to implement the instruction

\begin{verbatim}
MOVE rj, ri; // rj<- ri
\end{verbatim}

First give an intuitive explanation, then write your solution in pseudocode, not microcode (however each step must be readily implementable in microcode).

Intuition: Adjust the PC to point to the high-order byte of the instruction. Use this as the address to retrieve that byte from memory into the MAR. Mask and shift to extract the ri bits, storing those in a register. Repeat.

Pseudocode: Similarly, extracting the rj bits from the low-order byte, storing them in a register. Shift and combine these into a new instruction, with ri and rj switched, storing this to some available memory location (or overwriting the original instruction). Run this instruction as MOVE ri, rj.