What is your name?: ________________  (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 60 minutes.

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

T F  1. Stack frames in a recursive program must all be the same size.
T F  2. Addition using numbers of opposite signs never results in overflow.
T F  3. The stack in SPIM grows towards address 0.
T F  4. In a recursive program, every subprogram must store $ra on the stack.
T F  5. Everything you do with a branch instruction you can also do with a jump instruction.
T F  6. A jump-table in SPIM is used to implement the high-level construct of a switch-case statement. Each entry in the jump table must be the same size.
T F  7. For recursive programs to work properly in SPIM, parameter values must be stored and retrieved by the calling part of the program.
T F  8. The same arithmetic circuit can be used to compare integers and IEEE 754 floating point numbers.
T F  9. The PC is normally represented by $r7 in Mythsim.
T F  10. A carry-out at the most significant bit after an addition of two signed numbers always indicates overflow. No. It occurs when doing 2's comp. subtraction, e.g. 5-3.
T F  11. There are some values between 255 and 256 that cannot be represented using IEEE 764 single-precision floating point numbers.
T F  12. The key to be able to store very small values in IEEE 754 single-precision floating point format in denormalized form is that there is no implicit 1.
T F  13. Having a CDR (Control Data Register) speeds up the execution of a control program.
T F  14. For the binary multiplier circuit discussed in class, $n-1$ shifts are needed to implement multiplication of an $n$ bit number. $n$ shifts are needed.
T F  15. In Mythsim an instruction in memory can specify the operation: $r2 <- r3 + r4$
T F  16. The following two lines are equivalent:

```assembly
r2_write, a_sel0, a_sell, alu_sel=0;
r2_write, a_sel=3;
```

CS 366 - Computer Architecture II, Midterm Exam #2, page 1 of 5
17. In class we saw how the 8-step instruction fetch in Mythsim could actually be implemented in only 5 steps with no additional resources needed. An additional register (r6) is needed.

18. We can write microcode in Mythsim to explicitly set a register to some value, without using memory at all.

19. In Mythsim the results of an operation can be written to multiple registers at once.

20. If for some reason we started running out of registers, we could use the MDR briefly as a temporary storage location.

III. Short Answer: (6 points each)

1. Consider the code given below. What does this code do? (Don’t give an explanation for each line, but rather give the one-line summary that would be appropriate to use as the one line of documentation describing these two lines in a program.)

```
xor $t1, $t2, $t3
bgez $t1, label3
```

Branch to label3 if $t2 and $t3 have the same sign (either pos. or neg).

2. Briefly list and explain the two main kinds of hazards that can arise when using a pipelined implementation of SPIM.

   a) Data hazard - a register is used before a previous instruction has finished changing its value.

   b) Control hazard - the instruction after a branch is executed as part of the branch since it has already entered the pipeline before the branch instruction finishes executing.

3. Why does a pipelined implementation need a separate instruction cache and data cache?

   So that it can overlap writing results to memory with simultaneous reading instructions from memory.
4. Consider the following IEEE 754 single precision floating-point format number, represented in hexadecimal. What is the equivalent value as a decimal number?

\[ \begin{array}{c}
\text{4E30000} \\
\hline
0100 0011 1100 1100 \\
0100 0011 1110 0001 \\
4 3 \\
128 +7 \\
125 = 127 + 8 \\
\end{array} \]

\[ \begin{array}{c}
1.110011 \times 2^8 = 110000110 \\
3 \text{ pts.} \quad 2 \text{ pts.} \quad 1 \text{ pt.} \\
454 \\
\end{array} \]

5. Consider the following code:

1. \text{lw } \$t4, 0(\$t1) \quad \text{noop}
2. \text{add } \$t5, \$t1, \$t4
3. \text{xor } \$t6, \$t3, \$t4

Make an additions, changes, or deletions in the code above (if any) so that this code would run correctly in a pipeline implementation.

The \textit{noop} allows the write to \$t4 to finish in line 1 before \$t4 is read in line 2.

6. Consider Figure 8-9 from our text, which is a hardwired control unit for the Binary Multiplier circuit. Assuming we are not yet done, to what state will we go to next after the \textit{Shift-dec} control line is set? Write your answer in the space below the diagram and circle it.

\[ \text{FIGURE 8-9} \]

Control Unit for Binary Multiplier Using a Sequence Register and a Decoder

\[ \text{Shift-dec set in state 10, after which we go to state 01 where MUL0 is set} \]
7. What are the 3 factors that go into determining how many bits there needs to be in a microprogram control word?

\[\begin{align*}
\text{a) } & \text{ The number of control instructions (states) dictates the number of address bits needed for branching,} \\
\text{b) } & \text{ The number of status lines used to determine ASM diagram branching determines the number of select bits.} \\
\text{c) } & \text{ The number of control signals needed determines the number of control bits.}
\end{align*}\]

\[
\begin{array}{cccc}
\text{a} & \text{Next} & \text{Sel} & \text{Control} \\
\hline
\text{p} & \text{q} & \text{r} & \text{s}
\end{array}
\]

8. Intuitively, how would you implement an ADDB instruction in Mythsim? The ADDB instruction would implement b_bus + cin.

\[\begin{align*}
\text{1. Set some register to } & \alpha \ (\text{e.g. } r^y \leftarrow \alpha) \\
\text{2. Pass the b_bus register through the ALU, adding it to the } & \text{above register + cin. (e.g. } a_{-sel} = 4, \ r_k_{-sel}, \ alu_{-sel} = \text{ADD, cin})
\end{align*}\]

\[\text{if the b_bus is selected from an instruction, otherwise explicitly choose the register here, e.g. } b_{-sel} = 5\]

9. What are the implications the the delay slot for a jal instruction in a pipelined implementation? For instance, consider the code:

\[\begin{align*}
\text{1 jal sqr} \\
\text{2 li } & \$a0, 5 \\
\text{3 move } & \$a0, \$v0 //\text{get ready to display answer}
\end{align*}\]

Normally jal stores the address of the next instruction into \#ra. In pipelined mode, however, the next instruction \(2\) is actually the branch delay slot, and is executed before the jump. Line \(2\) would get executed an extra time when returning.

To solve this, \#ra needs to be set not to the next instruction \(2\), but rather to the instruction after that: \(3\).

This would have implications for the syscall above, since \$a0 might change.
10. What operation does the following microcode implement? Write it as a register-transfer operation (e.g. $r7 \leftarrow r3$).

```
r0_write, a_sel=2, b_sel=4, alu_sel=SUB, cin,
   if cout then goto label1 else goto label0 endif;
label0:  r1_write, a_sel=3, b_sel=5, alu_sel=SUB, cin=0, goto next;
label1:  r1_write, a_sel=3, b_sel=5, alu_sel=SUB, cin=1, goto next;
next:
```

$$
\begin{align*}
&\begin{array}{c}
\text{r1} \leftarrow \text{r0} \\
\text{r2} \leftarrow \text{r3} - \text{r4} \\
\text{r5} \leftarrow \text{r4} \quad + \text{cin} \\
\text{r1} & \quad \text{which is the same as:} \\
\text{r1} & \leftarrow \text{r0} - \text{r3} + \text{r5} + \text{r4} + 1
\end{array}
\end{align*}
$$