Name _____

Computer Architecture

EE 4720

Midterm Examination 1.0.3

Friday, 22 October 2004, 10:40–11:30 CDT

Problem 1	_____	(30 pts)
Problem 2	_____	(15 pts)
Problem 3	_____	(25 pts)
Problem 4	_____	(30 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Problem 1: The routine below is to do a binary search on an array and return 1 if the item is found, 0 otherwise. Complete the MIPS program using the C++ code as a guide. [30 pts]

☐ The only allowable synthetic instruction is `nop`.

☐ Fill as many delay slots as possible. The order of the assembly code **does not** have to match the order of the C code.

```
#  unsigned int delta = size;  int pos = 0;
#  while( true ) {
#      delta = delta >> 1;
#      int next_pos = pos + delta;
#      int e = array[next_pos];
#      if( e == target ) return 1;
#      if( !delta ) return 0;
#      if( e < target ) pos = next_pos;
#    }
       # CALL VALUE: $a0,  array: Address of array. Each element is a WORD-SIZED int.
       # CALL VALUE: $a1,  size: Number of elements in array. A power of 2.
       # CALL VALUE: $a2,  target: Value to find.
       # RETURN VALUE: $v0,  1, if target in array; 0, if not in array.
       # Can modify registers $a0-a3, $t0-$t9
```

Problem 2: In MIPS and many other ISAs floating-point instructions read and write a FP register file while most other instructions read and write general-purpose (integer) registers. Suppose MIPS-I were modified so that FP instructions used the same registers as the integer instructions. The registers are still 32 bits and can be used in pairs by instructions operating on double-precision operands. The new version of MIPS is not compatible with MIPS-I, don't worry about that.

(*a*) For each MIPS-I instruction below indicate whether it will be retained in the new ISA, "KEEP", or will not be needed, "OMIT." Also add any new instructions that might be needed. If an instruction is kept but operates differently describe the difference. [15 pts]

☐ Indicate KEEP or OMIT.

☐ If omitted show an existing instruction that would be used in its place.

☐ If kept, explain how it would operate differently.

lwc1:

ldc1:

cvt.w.d:

cvt.d.w:

mfc1:

mtc1:

add.s: Keep, but operates on integer registers.

add.d:

c.lt.s:

c.le.d:

bc1t:

Hint: In `bc1t` *b is for branch, t is for true.*

(*b*) If your answer for `bc1t` above was "OMIT" describe below how the instruction can be kept (possibly modified). If your answer was "KEEP" explain how it can be omitted. This will affect other instructions, list them and explain how they are affected.

Problem 3: Using an interesting technique the program below returns the value of register X, where X is a number in register `$a0`. For example, if `$a0` held an 8 the program would return the value of `$8` (or `$t0` using the register name) in register `$v0`. Consider:

```
        jal copyreg
        addi $a0, $0, 8
        # At this point $v0 has contents of $t0.

        addi $v0, $t0, 0   # $t0 is $8, register number 8
        # At this point $v0 has contents of $t0.
```

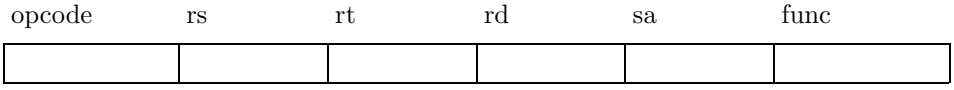
From the code above it looks like the `copyreg` routine is doing things the hard way. The one advantage is that it can copy a source register known only at run time, or that might change. The `addi` instruction always copies `$t0`. *Note: In the original exam the description and example above were not included.*

(*a*) Modify the program so that it takes a second call value, `a1`, which holds a second register number. The return value is put in that register. For example, if `a0` holds a 12 and `a1` holds a 3 then the routine copies the value in `r12` to `r3`. [17 pts]

☐ Show additions and changes. The added code must use a similar technique to the existing code.

Hint: Can be solved with two added instructions and a few minor modifications.

```
        # CALL VALUE:   $a0   A register number. (Except v0)


        # RETURN VALUE: $v0   The value in reg number X, where X is val in a0.

copyreg:
        la $v0, template    # Put address of label "template" into $v0.

        lw $v0, 0($v0)

        sll $a0, $a0, 21

        add $a0, $v0, $a0

        la $v0, mod

        sw $a0, 0($v0)

        jr $ra

mod:
        add $v0, $0, $0

template:
        add $v0, $0, $0
```



Format R: 31 26 25 21 20 16 15 11 10 6 5 0

Problem 3, continued:

(*b*) Suppose one were considering adding a new machine instruction, `copyreg`, that does the same thing as the routine above (before modification for part 1). Provide arguments for and against adding the new instruction. The arguments can include made-up data as long as it does not conflict with what has been given in this problem or in class. At least one of the arguments must refer to features of the code from part 1. [8 pts]

☐ Provide an argument in favor of adding the instruction.

☐ Provide an argument against adding the instruction. Do not simply reverse an assumption made above.

Problem 4: Answer each question below.

(*a*) A company is developing an implementation of an ISA. To guide their design they are using a **SUBSET** of the SPECcpu benchmark programs. [7 pts]

☐ Describe a situation in which that's a foolish thing to do and explain why it's foolish.

☐ Describe a situation in which that's a smart thing to do.

(*b*) Compiler optimizations are important. [8 pts]

☐ Describe a compiler optimization for which the compiler needs no knowledge of the ISA.

☐ Show an example.

☐ Describe a compiler optimization for which the compiler does need to know the target ISA.

(*c*) A goal in the design of MIPS and other RISC ISAs is to minimize the number of registers with specialized purposes. In MIPS `r31`, a.k.a. `$ra`, does have a specialized purpose. [7 pts]

☐ Why was it necessary to make an exception in this case?

(*d*) The target in MIPS branch instructions is specified by indicating the number of instructions to skip. [8 pts]

☐ Why couldn't such an approach be used in CISC ISAs?

☐ Specifying the number of instructions to skip saves two bits over what is needed for CISC targets. Why is saving two bits not as important in the design of CISC instructions?