Name Solution

Computer Architecture
EE 4720
Midterm Examination

Monday, 24 October 2005, 12:40–13:30 CDT

Problem 1		(50 pts)
Problem 2		(50 pts)
Exam Total		(100 pts)

Alias New Orleans Saints

Good Luck!

Problem 1: The partially completed routine on the next page is called with the address of a string which may contain characters that look like letters, for example, "`g0!1y`." The routine should replace those characters with the letters they look like; the example would be converted to "`golly`."

The string DMAP (see the next page) specifies the look-alikes. It specifies a letter (always lower case) followed by one or more look-alike characters followed by a comma (possibly followed by another group). For example, "`o0,l1!,`" indicates that the letter oh's look-alike is the digit zero and the letter el's look-alikes are the digit 1 and an exclamation point.

The routine on the next page is almost finished. The part at the end uses a look-up table to translate the string, and the part at the beginning has started setting up the look-up table. Write the code that finishes setting up the look-up table based on DMAP. [50 pts]

- ☑ Write the code that sets up the look-up table.
- ☑ The code must be reasonably efficient.
- ☑ The only synthetic instructions that can be used are `nop` and `la`.

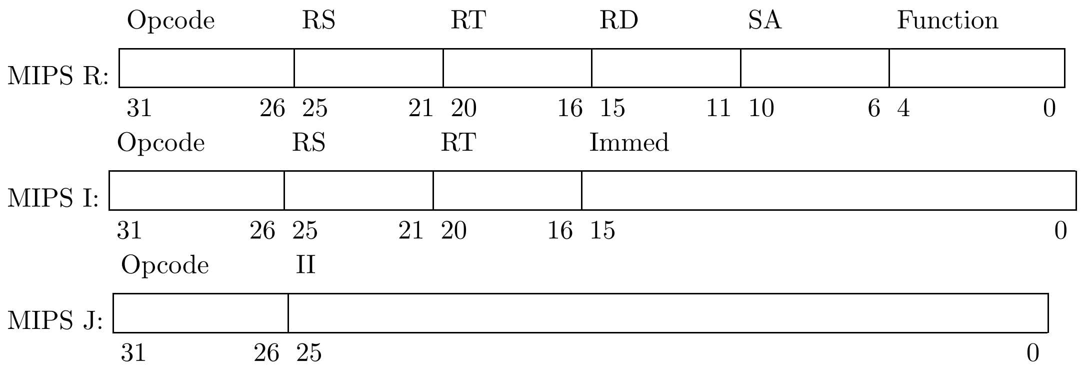
Solution on next page.

Grading Notes: To check for a comma in, say, `t4` many solutions used a `sub t3,` $t4$,`t5` followed by something like `beq` $t3$,`0, COMMA` (remember that `t5` held the comma character). The easier way to do it is just `beq` $t4$,`t5, COMMA`.

Use the next page for the solution.

```
        ## Register Usage
        #
        # $a0: Procedure call argument. Address of string to translate.
        # There are no return values.

DMAP:   .asciiz "o0,l1!,c([,t+,s$,"  # Translate 0->o, 1->l, !->l, (->c, etc.

LUT:    .space 256

        la $t0, LUT
        addi $t1, $0, 255

        # First, initialize look-up table so every character is mapped to itself.
        #
LOOP0:  add $t3, $t0, $t1         # Compute address of LUT entry.
        sb $t1, 0($t3)            # Write default entry. (A->A, B->B, etc.)
        bne $t1, $0  LOOP0
        addi $t1, $t1, -1

        # Next, set up look-up table based on DMAP string. (Finish in solution.)
        #
        la $t0, DMAP
        la $t1, LUT
        addi $t5, $0, 44   # ','  Comma character separates groups.

        # Start solution here. (Can be done in 9 insn.)

        # Solution Starts Here -- Solution Starts Here -- Solution Starts Here
LOOP1:
        lb $t4, 0($t0)
        beq $t4, $0 EXIT
        addi $t0, $t0, 1

LOOP2:
        lb $t3, 0($t0)
        beq $t3, $t5 LOOP1
        addi $t0, $t0, 1
        add $t3, $t3, $t1
        j LOOP2
        sb $t4, 0($t3)
EXIT:
        # Solution Ends Here -- Solution Ends Here -- Solution Ends Here


        # Use the look-up table to translate the string.
        la $t1, LUT
LOOP3:  lb $t2, 0($a0)       # Load character of string.
        beq $t2, $0, EXIT2
        add $t2, $t2, $t1    # Compute address of look-up table entry.
        lb $t2, 0($t2)       # Load translated character.
        sb $t2, 0($a0)       # Write translated character to string.
        j LOOP3
        addi $a0, $a0, 1

EXIT2:  jr $ra
        nop
```

Problem 2: Answer each question below. The instruction format descriptions are provided for reference.



(*a*) A proposed new MIPS branch instruction compares a register value to a constant to determine if the branch should be taken. For example, the branch below is taken if the contents of `t1` is 123. [10 pts]

```
beqi $t1, 123,  TARGET
nop
```

☑ Why isn't it feasible to code such an instruction using any of the existing MIPS formats?

Because that instruction uses two constants, the value to compare (`123`) and the branch target (`TARGET`). Since none of the three MIPS formats has two immediate fields it can't be coded. One could split the immediate field in Format I in half, use 8 bits for the value and 8 bits for the displacement, but then it would not be correct to say `beqi` used Format I. This is more than being picky about words because the fields in the format correspond to data paths in the implementation and so adding or modifying fields means adding or modifying hardware in the implementation.

☑ How could a conditional control transfer instruction that compared a register to an immediate be coded using the MIPS formats? (The instruction would be different than `beqi`.)

Put the branch target in a register. For example, `beqir t1, 123, t2`, in which the contents of `t1` is compared to `123`, if they are equal branch to the address in register `t2` (after the delay slot). Like `jr` and `jalr` the target address is in a register. This instruction would be coded in Format I: `beqir rs, immed, rt`.

☑ Show the instruction in assembly language.

See above.

(*b*) Show the results of addition for the 32-bit data types shown below. [10 pts]

✓ Unsigned Integer:

```
  0x0999
+ 0x0109
  0x0aa2
```

✓ BCD:

```
  0x0999
+ 0x0109
  0x1108
```

✓ Packed 4-bit integers with saturating arithmetic.:

```
  0x0999
+ 0x0109
  0x0a9f
```

(*c*) The MIPS ISA specifies that the sa (shift amount) field in the `add` instruction must be zero (an `add` instruction with a non-zero sa field value should cause an execution error). [10 pts]

✓ Describe a difficulty that might have arisen if the MIPS ISA had specified that implementations should ignore the sa field in an `add` instruction (and so the `sa` field could contain any value).

If implementations ignored the field then programmers would be free to put any value in the field without changing what the `add` does. Because programmers might do this a later version of the ISA could not use the field for an extended opcode or for an improved add instruction because existing code expects the `add` to be an add, regardless of the value in the field. As an extended opcode field, a non-zero value in sa wold specify completely different instructions, say a 1 in sa would specify a andn (and-not) instruction. An improved add instruction might use the sa field as a shift amount to be applied to the second operand (called a scaled add).

✓ Describe a difficulty that might have arisen if the MIPS ISA said nothing about the `sa` field in an add instruction.

The same difficulty as the previous case. If the ISA says nothing then some implementors would assume ignoring the field is okay and programmers for those implementations might put values in the field.

✓ Describe a difficulty that might have arisen if the MIPS ISA did specify that sa must be zero but did not say what should happen if the field were non-zero.

Not a very good ISA, since it's not fully describing the behavior of the `add` instruction. If some implementors chose to ignore non-zero values then there would be the same problem as the previous cases.

(*d*) Consider a benchmark suite that's similar to SPECcpu in that the tester is responsible for compiling the programs but that unlike SPEC, specifies that the tester should not use any optimizations when compiling the benchmarks. [10 pts]

☑ Compared to SPECcpu base and peak (result) scores, how useful would such results be? Explain.

Not very useful since there is no reason not to optimize code. A machine that is outperformed by its competitors on optimized code but outperforms its competitors on unoptimized code might be doing something clever, but also unnecessary since optimizing code is no big deal.

(*e*) Using a new compiler optimization a program's dynamic instruction count is cut in half. The values for CPI, IC, and ϕ are available for a run of the program compiled without the new optimization. Other than the instruction count, nothing has been measured for the program using the new optimization. How useful is the CPU performance equation for estimating the run time with the new optimization on the same system in this situation? Explain. [10 pts]

The CPU performance equation, $T = \frac{1}{\phi} \times \mathrm{IC} \times \mathrm{CPI}$, would not be very useful in estimating the run time. After the optimization we know IC (because that's given: half the previous value) and ϕ (its the same machine so it would be the same), but we don't know CPI. Since for a given machine CPI depends upon the instruction mix and arrangement we would expect them to change by a substantial amount with the new optimization because the new optimization had such a big effect on instruction count.