

CS61C Fall 2014 Discussion 3

1. Translate the following C code into MIPS.

<pre>// Strcpy: // \$s1 -> char s1[] = "Hello!"; // \$s2 -> char *s2 = // malloc(sizeof(char)*7); int i=0; do { s2[i] = s1[i]; i++; } while(s1[i] != '\0'); s2[i] = '\0';</pre>	<pre>addiu \$t0, \$0, 0 Loop: addu \$t1, \$s1, \$t0 # s1[i] addu \$t2, \$s2, \$t0 # s2[i] lb \$t3, 0(\$t1) # char is sb \$t3, 0(\$t2) # 1 byte! addiu \$t0, \$t0, 1 addiu \$t1, \$t1, 1 _____ _____ Done: sb \$t4, 1(\$t2)</pre>
<pre>// Nth_Fibonacci(n): // \$s0 -> n, \$s1 -> fib // \$t0 -> i, \$t1 -> j // Assume fib, i, j are these values int fib = 1, i = 1, j = 1; if (n==0) return 0; else if (n==1) return 1; n -= 2; while (n != 0) { fib = i + j; j = i; i = fib; n--; } return fib;</pre>	<pre>... _____ _____ _____ addiu \$s0, \$s0, -2 Loop: _____ addu \$s1, \$t0, \$t1 addiu \$t0, \$t1, 0 addiu \$t1, \$s1, 0 addiu \$s0, \$s0, -1 _____ Ret0: addiu \$v0, \$0, 0 j Done Ret1: addiu \$v0, \$0, 1 j Done RetF: addu \$v0, \$0, \$s1 Done: ...</pre>
<pre>// Collatz conjecture // \$s0 -> n unsigned n; L1: if (n % 2) goto L2; goto L3; L2: if (n == 1) goto L4; n = 3 * n + 1; goto L1; L3: n = n >> 1; goto L1; L4: return n;</pre>	<pre>L1: addiu \$t0, \$0, 2 div \$s0, \$t0 # puts (n%2) in \$hi mfhi \$t0 # sets \$t0 = (n%2) _____ j L3 L2: addiu \$t0, \$0, 1 _____ addiu \$t0, \$0, 3 mul \$s0, \$s0, \$t0 addiu \$s0, \$s0, 1 _____ L3: srl \$s0, \$s0, 1 _____ L4: ...</pre>

MIPS Addressing Modes

- We have several **addressing modes** to access memory (immediate not listed):
 - **Base displacement addressing:** Adds an immediate to a register value to create a memory address (used for lw, lb, sw, sb)
 - **PC-relative addressing:** Uses the PC (actually the current PC plus four) and adds the I-value of the instruction (multiplied by 4) to create an address (used by I-format branching instructions like beq, bne)
 - **Pseudodirect addressing:** Uses the upper four bits of the PC and concatenates a 26-bit value from the instruction (with implicit 00 lowest bits) to make a 32-bit address (used by J-format instructions)
 - **Register Addressing:** Uses the value in a register as a memory address (jr)

2. You need to jump to an instruction that $2^{28} + 4$ bytes higher than the current PC. How do you do it? Assume you know the exact destination address at compile time. (Hint: you need multiple instructions)

3. You now need to branch to an instruction $2^{17} + 4$ bytes higher than the current PC, when \$t0 equals 0. Assume that we're not jumping to a new 2^{28} byte block. Write MIPS to do this.

4. Given the following MIPS code (and instruction addresses), fill in the blank fields for the following instructions (you'll need your green sheet!):

```
0x002cff00: loop: addu $t0, $t0, $t0      | 0 |   |   |   |   |   |
0x002cff04:      jal  foo                    | 3 |   |   |   |   |   |
0x002cff08:      bne  $t0, $zero, loop          | 5 | 8 |   |   |   |   |
...
0x00300004: foo: jr $ra                      $ra=_____
```

5. What instruction is 0x00008A03?