

**Exercises for “Computer Science II” — SS 2003**  
**No. 4**  
Due: May 30, 2003

---

### The Assembler

Assemblers translate symbolic representations of instructions into machine code that can be executed by the CPU. For example, the instruction `add $t3,$t3,$t5` is translated by the assembler to the machine code whose hexadecimal representation is `0x016d5820`. The assembler performs this translation by finding the template for the instruction and inserting register numbers and/or constant values into the appropriate locations. For our simple `add` example, the assembler uses the `add` template found on page A-55 of <http://www.cs.wisc.edu/larus/SPIM/cod-appa.pdf>.

<code>add rd, rs, rt</code>	6 0	5 rs	5 rt	5 rd	5 0	6 0x20
-----------------------------	--------	---------	---------	---------	--------	-----------

For our purposes, it is convenient to convert the hexadecimal values in the template to binary which gives us:

6 000000	5 rs	5 rt	5 rd	5 00000	6 100000
-------------	---------	---------	---------	------------	-------------

The assembler knows that register `$t3` is number 11 and that register `$t5` is number 13 (see page A-23). Substituting these values into the template produces:

6 000000	5 01011	5 01101	5 01011	5 00000	6 100000
-------------	------------	------------	------------	------------	-------------

This is the binary encoding of the assembly language instruction `add $t3,$t3,$t5` that the assembler will emit into an object file. We can also convert the binary encoding of the instruction into hexadecimal which is more convenient for humans to read. As we have already stated, the hexadecimal encoding of the `add` in this example is `0x016d5820`.

1. (4 Points) Obtain a copy of <http://www.cs.wisc.edu/larus/SPIM/cod-appa.pdf>. Using the instruction templates beginning on page A-55 and the register numbers given on page A-23, give the binary and hexadecimal encodings for the following assembly language instructions.
  - `xor $t0,$t1,$t9`
  - `andi $s3,$t2,0x17e`
  - `sll $t3,$t3,9`
  - `j 0x03001020`

You may use SPIM to check your answers.

## Control Flow

High-level languages typically have several types of control structures. For example, the Java programming language has an `if` statement, along with `while`, `do`, and `for` loops. Assembly languages typically have only a single control structure, the branch or jump. Fortunately it is possible to emulate the semantics of any high-level language control structure using branches and jumps. For example, the statement

```
if (up == 1) {
    i = i + 1;
} else {
    i = i - 1;
}
```

can be implemented in MIPS assembly language as follows:

```
    bne $t1,1,L1    # If $t1 (up) isn't 1, branch to L1
    addi $t2,$t2,1  # $t1 (up) was one, so increment $t2 (i)
    b    L2         # Branch to end of if statement
L1:  addi $t2,$t2,-1 # $t1 (up) wasn't one, so decrement $t2 (i)
L2:  ...          # End of if statement
```

For the sake of simplicity, we assume that the variable `up` is held in register `$t1` and variable `i` is held in register `$t2`.

It is also relatively straightforward to implement loops in assembly language. For example, the `while` loop

```
i = 0;
j = 0;
while (i < 10) {
    j = j + 1;
    i = i + 1;
}
```

can be implemented in MIPS assembly language as follows:

```
    li $t1,0        # Set $t1 (i) to 0
    li $t2,0        # Set $t2 (j) to 0
L1:  bge $t1,10,L2  # If $t1 (i) >= 10, branch to L2
    addi $t2,$t2,1  # Add 1 to $t2 (j)
    addi $t1,$t1,1  # Add 1 to $t1 (i)
    b    L1         # Branch to beginning of loop
L2:  ...          # Loop finished
```

Again, for the sake of simplicity, we assume that the variable `i` is held in register `$t1` and variable `j` is held in register `$t2`.

2. (4 Points) Using the while loop code in the example as your guide, write the MIPS assembly language code for the loop

```
i = 20;
j = 0;
do {
    i = i - 1;
    j = j + 1;
} while (i >= 0);
```

You may assume that the variable `i` is represented by register `$t1` and the variable `j` by register `$t2`. You may check your work using SPIM.

3. (2 Points) Write the MIPS assembly language code for the loop

```
j = 0;
for(i=0;i<10;i++) {
    j = j + 1;
}
```

Again, you may assume that the variable `i` is represented by register `$t1` and the variable `j` by register `$t2`. You may check your work using SPIM.