# Computer Architecture – Assembly Homework

**Exercise 1)** Explain how the following program can be used to determine whether a computer is big-endian or little-endian:

```
li $t0, 0x6789CDBA
sw $t0, 100($0)
lb $s5, 101($0)
```

**Exercise 2)** Write the following strings using ASCII encoding. Write your final answers in hexadecimal.

. (a) ABBA
. (b) Perfect 20!
. (c) (your own name)

**Exercise 3)** Show how the strings in Exercise 2 are stored in a byte-addressable memory on (a) a big-endian machine and (b) a little-endian machine starting at memory address 0x1000100C. Use a memory diagram. Clearly indicate the memory address of each byte on each machine.

**Exercise 4)** Convert the following MIPS assembly code into machine language. Write the instructions in hexadecimal.

```
addi $s0, $0, 53
sw $t1, -7($t2)
sub $t1, $s7, $s2
```

Which register contents before the beginning of the program do the results depend on?

What will be the register or memory values that have changed after the end of the program?

Which instructions from Exercise 6.8 are I-type instructions? Sign-extend the 16-bit immediate of each such instruction so that it becomes a 32-bit number.

**Exercise 5)** Convert the following program from machine language into MIPS assembly language. The numbers on the left are the instruction address in memory, and the numbers on the right give the instruction at that address. Then reverse engineer a high-level program that would compile into this assembly language routine and write it. Explain in words what the program does. $a0 is the input, and it initially contains a positive number, n. $v0 is the output.

```
0x00400000 0x20080000
0x00400004 0x20090002
0x00400008 0x0089502a
0x0040000c 0x15400003
0x00400010 0x01094020
0x00400014 0x21290004
0x00400018 0x08100002
0x0040001c 0x01001020
```

Exercise 6) Write a procedure in a high-level language for int findsum64(int array[], int size). size specifies the number of elements in the array. array specifies the base address of the array.

The procedure should return the index number of the first array entry the sum of the numbers before it (including itself) surpasses 64. If this does not happen, it should return the value -1. After writing in a high-level language (C), translate to assembly.

**Exercise 7)** Consider the following MIPS assembly language snippet. The numbers to the left of each instruction indicate the instruction address.

```
0x00400028 add $a0, $a1, $0
0x0040002c           jal f2
0x00400030 f1:   jr $ra
0x00400034 f2:   sw $s0, 0($s2)
0x00400038           bne $a0, $0, else
0x0040003c           j f1
0x00400040 else:addi $a0, $a0, 02
0x00400044           j f2
```

(a) Translate the instruction sequence into machine code. Write the machine code instructions in hexadecimal.
(b) List the addressing mode used at each line of code.

**Exercise 8)** Consider the following high-level procedure.

```
// high-level code
int f(int n, int k) {
        int b;
        b = k + 3;
        if(n == 0) b = 11;
        else b = b + (n * n) + f(n - 1, k + 1); return b * k;
}
```

(a) Translate the high-level procedure f into MIPS assembly language. Pay partic- ular attention to properly saving and restoring registers across procedure calls and using the MIPS preserved register conventions. Clearly comment your code. You can use the MIPS mult, mfhi, and mflo instructions. The procedure starts at instruction address 0x00400200. Keep local variable b in $s0.

(b) Step through your program from part (a) by hand for the case of f(2, 4). Draw a picture of the stack similar to the one in Figure 6.26(c). Write the register name and data value stored at each location in the stack and keep track of the stack pointer value ($sp). You might also find it useful to keep track of the values in $a0, $a1, $v0, and $s0 throughout execution. Assume that when f is called, $s0 = 0xABCD and $ra = 0x400004. What is the final value of $v0?