Chapter 2 Questions - Assignment Questions for Week 2

2.1 ¡§2.2¿ For the following C statement, write the corresponding RISC-V assembly code. Assume that the C variables `f`, `g`, and `h`, have already been placed in registers `x5`, `x6`, and `x7` respectively. Use a minimal number of RISC-V assembly instructions.

```
f = g + (h - 5);
```

2.2 ¡§2.2¿ Write a single C statement that corresponds to the two RISC-V assembly instructions below.

```
add f, g, h
add f, i, f
```

2.3 ¡§§2.2, 2.3¿ For the following C statement, write the corresponding RISC-V assembly code. Assume that the variables `f`, `g`, `h`, `i`, and `j` are assigned to registers `x5`, `x6`, `x7`, `x28`, and `x29`, respectively. Assume that the base address of the arrays `A` and `B` are in registers `x10` and `x11`, respectively.

```
B[8] = A[i - j];
```

2.4 ¡§§2.2, 2.3¿ For the RISC-V assembly instructions below, what is the corresponding C statement? Assume that the variables `f`, `g`, `h`, `i`, and `j` are assigned to registers `x5`, `x6`, `x7`, `x28`, and `x29`, respectively. Assume that the base address of the arrays `A` and `B` are in registers `x10` and `x11`, respectively.

```
slli x30, x5, 3 // x30 = f*8
add x30, x10, x30 // x30 = &A[f]
slli x31, x6, 3 // x31 = g*8
add x31, x11, x31 // x31 = &B[g]
ld x5, 0(x30) // f = A[f]
addi x12, x30, 8
ld x30, 0(x12)
add x30, x30, x5
sd x30, 0(x31)
```

2.5 ¡§2.3¿ Show how the value `0xabcdef12` would be arranged in memory of a little-endian and a big-endian machine. Assume the data is stored starting at address 0 and that the word size is 4 bytes.

2.7 ¡§§2.2, 2.3¿ Translate the following C code to RISC-V. Assume that the variables `f`, `g`, `h`, `i`, and `j` are assigned to registers `x5`, `x6`, `x7`, `x28`, and `x29`, respectively. Assume that the base address of the arrays `A` and `B` are in registers `x10` and `x11`, respectively. Assume that the elements of the arrays `A` and `B` are 8-byte words:

```
B[8] = A[i] + A[j];
```

2.10 ¡§2.4¿ Assume that registers `x5` and `x6` hold the values `0x8000000000000000` and `0xD000000000000000`.

2.10.1 What is the value of `x30` for the following assembly code?

```
add x30, x5, x6
```

2.10.2 Is the result in `x30` the desired result, or has there been overflow?

2.10.3 For the contents of registers `x5` and `x6` as specified above, what is the value of `x30` for the following assembly code?

```
sub x30, x5, x6
```

2.10.4 Is the result in `x30` the desired result, or has there been overflow?

2.10.5 For the contents of registers `x5` and `x6` as specified above, what is the value of `x30` for the following assembly code?

```
add x30, x5, x6
add x30, x30, x5
```

2.10.6 Is the result in `x30` the desired result, or has there been overflow?

2.12 ¡§§2.2, 2.5¿ Provide the instruction type and assembly language instruction for the following binary value (Hint: Figure 2.20 may be helpful):

```
0000 0000 0001 0000 1000 0000 1011 0011 (two)
```

2.13 ¡§§2.2, 2.5¿ Provide the instruction type and hexadecimal representation of the following instruction:

```
sd x5, 32(x30)
```

2.16 ¡§§2.5, 2.8, 2.10¿ Assume that we would like to expand the RISC-V register file to 128 registers and expand the instruction set to contain four times as many instructions.

2.16.1 How would this affect the size of each of the bit fields in the R-type instructions?

2.16.2 How would this affect the size of each of the bit fields in the I-type instructions?

2.16.3 How could each of the two proposed changes decrease the size of a RISC-V assembly program? On the other hand, how could the proposed change increase the size of an RISC-V assembly program?

2.22 ¡§2.10¿ Suppose the program counter (PC) is set to 0x20000000.

2.22.1 What range of addresses can be reached using the RISC-V jump-and-link (jal) instruction? (In other words, what is the set of possible values for the PC after the jump instruction executes?)

2.22.2 What range of addresses can be reached using the RISC-V branch if equal (beq) instruction? (In other words, what is the set of possible values for the PC after the branch instruction executes?)

2.23 ¡§2.7, 2.10¿ Consider a proposed new instruction named `rpt`. This instruction combines a loop's condition check and counter decrement into a single instruction. For example `rpt x29`, loop would do the following:

```
if (x29 > 0) {
x29 = x29 -1;
goto loop
}
```

2.23.1 If this instruction were to be added to the RISC-V instruction set, what is the most appropriate instruction format?

2.23.2 What is the shortest sequence of RISC-V instructions that performs the same operation?

2.27 ¡§2.7¿ Translate the following loop into C. Assume that the C-level integer i is held in register `x5`, `x6` holds the C-level integer called result, and `x10` holds the base address of the integer MemArray.

```
addi x6, x0, 0
addi x29, x0, 100
LOOP: ld x7, 0(x10)
add x5, x5, x7
addi x10, x10, 8
addi x6, x6, 1
blt x6, x29, LOOP
```

2.28 ¡§2.7¿ Rewrite the loop from Exercise 2.27 to reduce the number of RISC-V instructions executed. Hint: Notice that variable i is used only for loop control.

2.36 ¡§2.10¿ Write the RISC-V assembly code that creates the 64-bit constant 0x1122334455667788two and stores that value to register x10.

2.40 ¡§§1.6, 2.13¿ Assume that for a given program 70% of the executed instructions are arithmetic, 10% are load/store, and 20% are branch.

2.40.1 Given this instruction mix and the assumption that an arithmetic instruction requires two cycles, a load/store instruction takes six cycles, and a branch instruction takes three cycles, find the average CPI.

2.40.2 For a 25% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?

2.40.3 For a 50% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?