CSE3666

Homework 3

Due Date: By the end of Monday, 02/21/2022.
Total points: 200

Check your answers to Problems 1 to 4 in HW3-test in HuskyCT.
Submit your answers to Problems 5 and 6 in a PDF file in HuskyCT.

Always explain your code with concise comments.

1. Encoding. For each RISC-V instruction, find out its encoding format, *the bits* in each field, and the machine code as 8 hexadecimal digits. An example is shown below. Pay attention to the number of bits in each field.

```
or   s1, s2, s3
slli t1, t2, 16
xori x1, x1, -1
lw   x2, -100(x3)
```

Example:

```
Instruction: addi  t0, t1, 3
Find out register numbers: addi x5,x6,3
I-type
Immediate: 000000000011
opcode:   0010011
rd:       00101
funct3:   000
rs1:      00110
rs2:      00011
funct7:   0000000
machine code in bits: 00000000001100110000001010010011
machine code in hex: 00330293
```

2. Decoding. Each 8-digit hexadecimal number in the following table represents a RISC-V instruction. For each machine code, find its encoding format, bits in each field, and then decode it into a RISC-V instruction. The steps are the reverse of those in Problem 1. Use register numbers (like x0 instead of zero). Any immediate or displacement (offset) should be in decimal.

   When entering instructions in HW3-test, keep only one space after instruction name. For example, there is only one space in "`lw x2,-100(x3)`". There is no space after the comma and around the parentheses.

	Machine Code
a	`0xfeaca823`
b	`0x04020713`
c	`0x00557bb3`
d	`0x414fdf13`

3. Find the machine code for the following instruction. Assume all instructions are labeled sequentially, for example, I1, I2, I3, …, I200.

```
I10 :         BGE   x10, x20, I100
I11 :         BEQ   x10, x0, I1
```

4. Decode the following instructions in machine code. Find the offset and then the target address. The hexadecimal number before the colon is the instruction's address.

```
0x0400_366C:            0xDB5A_04E3
```

5. Translate function f() in the following C code to RISC-V assembly code. Assume function g has already been implemented. The constraints are:
    1) Allocate register s1 to sum, and register s2 to i.
    2) Save registers at the beginning of the function and restore them before the exit.
    3) There are no load or store instructions in the loop. If we want to preserve values across function calls, place the value in a saved register before the loop. For example, we keep variable i in register s2.

   Your code should follow the flow of the C code. Write concise comments. Clearly mark instructions for saving registers, loop control, function, restoring register, and so on.

   Reminder: the callee can change any temporary and argument registers.

```
// prototype of g
// the first argument of g is an address of an integer
int  g(int * a, int i);

int f(int d[1024])
{
    int sum = 0;
    for (int i = 0; i < 1024; i += 1) {
        sum += g(&d[i], i);       // pass d[i]'s address to g
    }
    return sum;
}
```

6. Translate function msort() in the following C code to RISC-V assembly code. Assume merge() and copy() are already implemented. The array passed to msort() has at most 256 elements.

   Your code should follow the flow of the C code. Write concise comments. Clearly mark instructions for saving registers, function calls, restoring register, and so on.

   To make the code easier to read, we can change sp twice at the beginning of the function: once for saving registers and once for allocating memory for array c.

   The function should have only one exit. There is only one return instruction.

   Another reminder: the callees can change any temporary and argument registers.

```
void merge(int c[], int d1[], int n1, int d2[], int n2);
void copy(int d[], int c[], int n);

void  msort(int d[], int n)
{
    int c[256];

    if (n <= 1)
        return;

    int n1 = n / 2;

    msort(d, n1);
    msort(&d[n1], n - n1);    // &d[n1] means the address of d[n1]
    merge(c, d, n1, &d[n1], n - n1);
    copy(d, c, n);
}
```