2021 Computer Architecture Problem Set #2

※ If there is no specified data type, you can just assume that 1 variable or 1 array element size is 8-byte.

double word.

1. Translate the following RISC-V code to C. Assume that the variables f is assigned to registers x5. Assume that the base address of the array A is in register x10.

```
addi x30, x10, 8      x30 ← &A[1]
addi x31, x10, 0      x31 ← &A[0]
sd x31, 0(x30)        A[1] = &A[0]
ld x30, 0(x30)        x30 ← &A[0]
add x5,x30, x31
```

x5 ← f, x10 ← A

f = &A[0] + &A[0] → f = 2*(&A)

2. For the RISC-V assembly instructions below, what is the corresponding C statement(s)? Assume that the variables f is assigned to register x5. Assume that the base address of the arrays A and B are in registers x10 and x11, respectively.

```
slli x30, x5, 3       8f
add x30, x10, x30     x30 ← &A[f]
slli x31, x6, 3
add x31, x11, x31     x31 ← &B[g]
ld x5, 0(x30)         x5 ← A[f]
addi x12, x30, 8      x12 ← &A[f+1]
ld x30, 0(x12)        x30 ← A[f+1]
add x30, x30, x5      x30 ← A[f+1] + A[f]
sd x30, 0(x31)
```

x5 ← f, x10 ← A, x11 ← B

B[g] = A[f+1] + A[f]

3. Consider the following RISC-V loop:

```
LOOP:   beq x6, x0, DONE
        addi x6, x6, -1
        addi x5, x5, 2
        jal x0, LOOP
DONE:
```

Assume that the register x6 is initialized to the value 10. What is the final value in register x5 assuming the x5 is initially zero?

x6 ← 10, 9, 8, …, 0
x5 ← 0, 2, 4, …
= 20

4. Translate the following loop into C code. 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
```

x5 ← i
x6 ← result ← 0
x10 ← MemArray의 Base 주소.
x29 ← 100
x7 ← MemArray[0]
MemArray[i].

```
addi x6, x6, 1
blt x6, x29, LOOP
```

x6 < 100.

for (j=0; j<100; j++){
i = i + MemArray[j];
}

x5 x10

5. Translate the following C code to RISC-V assembly codes. Assume that the variables i and j are assigned to registers x5 and x6, 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];

x5 ← i, x6 ← j, x10 ← Base of "A", x11 ← Base of B", x28 ← A[i]

```
slli x5, x5, 3
add  x28, x10, x5
ld   x29, 0(x28)
slli x6, x6, 3
add  x28, x10, x6
ld   x30, 0(x28)
add  x30, x30, x29
sd   x30, 64(x11)
```

6. For the following C statement, write the corresponding RISC-V assembly code. Assume that the variables i and j are assigned to registers x5 and x6, respectively. Assume that the base address of the arrays A and B are in registers x10 and x11, respectively.

B[8] = A[i−j];

```
sub  x7, x5, x6      // x7 = i-j
slli x7, x7, 3       // x7 = 8(i-j)
add  x28, x10, x7    // base of A + x7 =
ld   x29, 0(x28)     // x29 = A[i-j]
sd   x29, 64(x11)    // B[8] = A[i-j]
```

7. Translate the following C code to RISC-V assembly code. Assume that the values of a, b, i, and j are in registers x5, x6, x7, and x29, respectively. Also, assume that register x10 holds the base address of the array D.

for(i=0; i<a; i++)

for(j=0; j<b; j++)

D[4*j] = i + j;

D는 32씩 증가 → j++

```
LOOPI:
  addi x7, x0, 0       i=0
  bge  x7, x5, ENDI    while i<a
  addi x30, x10, 0     x30 = &D
  addi x29, x0, 0      j=0
LOOPJ:
  bge  x29, x6, ENDJ   while j<b
  add  x31, x7, x29    x31 = i+j
  sd   x31, 0(x30)     D[4*j] = i+j
  addi x30, x30, 32    x30 = &D[4*(j+1)]
  addi x29, x29, 1     j++
  jal  x0, LOOPJ
ENDJ:                  for 다음 코드
  addi x7, x7, 1       i++
  jal  x0, LOOPI
ENDI:
```

8. Suppose the program counter (PC) is set to 0x20000000. 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?) 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?)

jal → UJ type → Immediate (12+1−2) → 2^{12}
beq → SB type → Immediate (20+1−2) → 2^{18}

beq instruction [0x1FFFF000, 0x20000FFE] → $-2^{12} \sim 2^{12}-1$

jal instruction [0x1FF00000, 0x200FFFFE] → $-2^{20} \sim 2^{20}-1$

Instruction의 범위가

9. Assume the following register contents: x5 = 0x00000000AAAAAAAA, x6 = 0x1234567812345678. For the register values shown above, what is the value of x7 for the following sequence of instructions?

0x12345678BABEFEF8

slli x7, x5, 4 — not sign extension

or x7, x7, x6

00000000	00000000	00000000	00000000	10101010	10101010	10101010	10101010
00000000	00000000	00000000	00001010	10101010	10101010	10101010	10100000
00010010	00110100	01010110	01111000	00010010	00110100	01010110	01111000
00010010	00110100	01010110	01111010	10111010	10111110	11111110	11111000

10. Translate the following C code into RISC-V assembly language. You can assume the registers that correspond to the C variables.

temp= v[k];

v[k]= v[k+1];

v[k+1] = temp;

base of array v → x8, k → x5, temp → x6

```
slli x5, x5, 3      // 8×k
add  x7, x8, x5     // base of v + x5
ld   x28, 0(x7)     // x28 ← v[k]
add  x6, x0, x28    // temp = v[k]
addi x29, x5, 8     // x29 ← 8×k+8 = 8(k+1)
add  x30, x8, x29   // x30 = base of v + x29
ld   x31, 0(x30)    // x31 = v[k+1]
sd   x31, 0(x7)     // v[k] = v[k+1]
sd   x6, 0(x30)     // v[k+1] = temp
```

11. Translate function f into RISC-V assembly language. Assume the function declaration for g is **long long int g(long long int a, long long int b)**. The code for function f is as follows:

long long int f(long long int a, long long int b, long long int c, long long int d){

return g(g(a,b), c+d);

}

x10 x11 x12 x13

x10, x11 그대로 g 호출

저장해 놔야 함

2칸 할당해야 함

```
f:
  addi x2, x2, -16
  sd   x1, 0(x2)
  add  x5, x12, x13
  sd   x5, 8(x2)
  jal  x1, g          // g(a,b) 호출
  ld   x11, 8(x2)
  jal  x1, g          // g(g(a,b), c+d)
  ld   x1, 0(x2)      → return address 복구
  addi x2, x2, 16     → stack pointer 복구
  jalr x0, 0(x1)
```

0x20000000 ± 2^{10}

- 0x00000200 ... -2^{10}

-2^{10}

0010 0000 0000 0000 0000 0000 0000 0000

$+$ 0000 0000 0000 0000 0001 0000 0000 0000

1111 1111 1111 1111 1110 1111 1111 1111

$+$ 1 1

1111 1111 1111 1111 11111 0000 0000 0000 0

0001 1111 1111 1111 1111 0000 0000 0000

0x1FFFF000 ~ 0x20000FFF

$\pm 2^{10}$

0x2000 0000

$-$0x2002 0000

0010 0000 0000 0000 0000 0000 0000 0000

$-$ 0000 0000 0000 0010 0000 0000 0000 0000

1111 1111 1111 1101 1111 1111 1111 1111

$+$ 1

1111 1111 1111 1110 0000 0000 0000 0000 0

~~1~~ 0001 1111 1111 1110 0000 0000 0000 0000

0x1FFE0000 ~ 0x10