Tutorial 2 – Week of Sep 27

Q1) 2.1

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);

Solution

```
addi    x5, x7, -5      # f= h - 5
add     x5, x5, x6      # f= g + f
```

(note, no subi)

Q2) 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];

we don't need variable f,g,h, so we should remove f,g,h and x5,x6,x7

Solution

64-bit

```
        sub     x30, x28, x29       // compute i-j
        slli    x30, x30, 3         // multiply by 8 to convert the work offset to a byte offset
        add     x3, x3, x30
        ld      x30, 0(x3)          // load A[i-j]
        sd      x30, 64(x11)        // Store in B[8]
```

x3 should be x10

it is typo of "word" but I think using "index" should be better. (using word can be confusing when talking about 32 bits and 64 bits)

32-bit

```
        sub     x30, x28, x29       // compute i-j
        slli    x30, x30, 2         // multiply by 4 to convert the work offset to a byte offset
        add     x3, x3, x30
        lw      x30, 0(x3)          // load A[i-j]
        sw      x30, 32(x11)        // Store in B[8]
```

Q3) 2.7

we don't need f, g, h, x5, x6, x7

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];

Solution

I think this solution is not right, because we changed the value of x11(base address of B) before storing.
And also it is calculating B[8] = A[i] + B[j]

64-bit

```
slli    x28, x28, 3          # x28 = i *8
add     x10, x10, x28        #
ld      x28, 0(x10)          # x28 = A[i]
slli    x29, x29, 3          # x29 = j*8
add     x11, x11, x29        # x11 address of B[j]
ld      x29, 0(x11)          # x29 = B[j]
sd      x29, 64 (x11)        # Store result in B[8]
```

32-bit

```
slli    x28, x28, 2          # x28 = i *4
add     x10, x10, x28        #
lw      x28, 0(x10)          # x28 = A[i]
slli    x29, x29, 2          # x29 = j*4
add     x11, x11, x29
lw      x29, 0(x11)          # x29 = B[j]
sw      x29,  32(x11)        # Store result in B[8]
```

my solution

64-bit

```
slli x28, x28, 3 # x28 = i * 8
add x28, x10, x28 # x28 = &A[0] + i * 8
ld x28, 0(x28) # x28 = A[i]
slli x29, x29, 3 # x29 = j * 8
add x29, x10, x29 # x29 = &A[0] + j * 8
ld x29, 0(x29) # x29 = A[j]
add x28, x28, x29 # x28 = A[i] + A[j]
sd x28, 64(x11) # store result in B[8]
```

32-bit

```
slli x28, x28, 2 # x28 = i * 4
add x28, x10, x28 # x28 = &A[0] + i * 4
lw x28, 0(x28) # x28 = A[i]
slli x29, x29, 2 # x29 = j * 4
add x29, x10, x29 # x29 = &A[0] + j * 4
lw x29, 0(x29) # x29 = A[j]
add x28, x28, x29 # x28 = A[i] + A[j]
sd x28, 32(x11) # store result in B[8]
```