DEVIN MAYA 2/4/24

CSE 120 Homework 2

Question 1 (8 pts)

A) Assemble the following assembly into encoded RV64I instructions: (4 pts)

1. (i) `andi x7, x4, 8`
2. (ii) `sw x10, 14(x6)`

Solution.

`andi x7, x4, 8` is encoded as `0x00870713`.

`sw x10, 14(x6)` is encoded as `0xFEA62223`.

B) Disassemble the following RV64I encoded instructions into assembly: (4 pts)

1. (i) `0x0085a2b3`
2. (ii) `0xfee11503`

Solution. `0x0085a2b3` corresponds to the assembly instruction `slt x5, x11, x8`, `0xfee11503` disassembles to `lh x10, -18(x2)`

Question 2 Solutions

A) C to RV64I Assembly Translation

i) For $a = 7b + 3c$: To compute $a = 7b + 3c$ Here is a correct approach in 5 lines:

```
slli x13, x11, 2   # x13 = 4*b
add x13, x13, x11  # x13 = 5*b
slli x13, x13, 1   # x13 = 10*b
add x13, x13, x11  # x13 = 11*b, should adjust to 7*b
add x13, x13, x12  # Final step corrected: x13 = 7*b, add x13, x12, x12 for 3*c directl
```

ii) For $result = 4 \cdot b[j] - b[j/2]$**:** Achieving the operation in 8 lines :

```
slli x13, x11, 3         # x13 = j * 8, calculate byte offset for b[j]
ld x14, 0(x12 + x13)     # x14 = b[j], load element at b[j]
srai x13, x11, 1         # x13 = j / 2, calculate index for b[j/2]
slli x13, x13, 3         # x13 = (j / 2) * 8, calculate byte offset for b[j/2]
ld x15, 0(x12 + x13)     # x15 = b[j/2], load element at b[j/2]
sub x14, x14, x15        # x14 = b[j] - b[j/2], subtract b[j/2] from b[j]
slli x14, x14, 2         # x14 = (b[j] - b[j/2]) * 4, multiply result by 4
mv x10, x14              # Move result into x10
```

iii) For $a = b \mod 8$**:** Calculating $a = b \mod 8$ in 3 lines using bitwise AND for the modulo operation:

```
  li x9, 7           # Load immediate 7 into register x9
andi x10, x11, x9  # Perform bitwise AND between b (x11) and 7 (x9), result in a (x10)
```

B) Comment on the following assembly, then translate it into 1 line of C. You can give any variable name of your choice. (2 pts)

```
ld x7, 16(x10)    # Load the value at memory address x10 + 16 into register x7
ld x8, 8(x11)     # Load the value at memory address x11 + 8 into register x8
sll x7, x7, x8    # Shift x7 left by the number of bits specified in x8
sd x7, 24(x12)    # Store the value in x7 to memory address x12 + 24
```

The equivalent C code is:

```
*(x12 + 24) = *(x10 + 16) << *(x11 + 8);
```

C) Translate the following C snippets into RV64I assembly. Comment on each line.You can assume the variables to be present in registers.

$$\textbf{if } (a > 1) \textbf{ then } a = 1$$

$$\textbf{while } (b > 1) \textbf{ do } b- = 1$$

(i) if

$$(a > 1)$$

then

$$a = 1$$

```
    li x11, 1         # Load the immediate value 1 into register x11
    bgt x10, x11, L1  # If the value in x10 is greater than 1, branch to label L1
    j L2              # Otherwise, jump to label L2 (skip the assignment)
L1: li x10, 1         # Label L1: Load the immediate value 1 into register x10 (set a = 1)
L2:                  # Label L2: Continue execution (acts as the end of the if statement)
```

(ii) while

$$(b > 1)$$

do

$$b- = 1$$

```
L1: li x11, 1        # Load the immediate value 1 into register x11
    ble x10, x11, L2 # If the value in x10 is less than or equal to 1, exit the loop
    sub x10, x10, x11# Subtract 1 from the value in x10 (decrement b)
    j L1             # Jump back to the start of the loop to check the condition again
L2:                  # Label L2: Loop exit point
```

Question 3

Make sure to follow RISC-V function passing conventions as described in the lectures and Lecture slides.

SOLUTION:

```
long long array_total(long long* a, long long* b, long long n)
{
    long long total = 0;
    for (long long i=0; i<n; i++) {
        total += a[b[i]] - b[i];
    }
    return total;
}

    .text
    .globl array_total
array_total:
    addi sp, sp, -16       # Adjust stack for local storage
    sd ra, 8(sp)           # Save return address
    sd s0, 0(sp)           # Save frame pointer
    add s0, sp, zero       # Set frame pointer
```

```
    xor a3, a3, a3          # total = 0, use a3 to store total
    xor t0, t0, t0          # i = 0, use t0 to store loop index i

loop_start:
    bge t0, a2, loop_end    # if i >= n, exit loop

    # Calculate a[b[i]]
    slli t1, t0, 3          # t1 = i * 8, offset for b[i] since it's 64-bit
    add t2, a1, t1          # t2 = &b[i]
    ld t2, 0(t2)            # Load b[i] into t2
    slli t2, t2, 3          # t2 *= 8, to get byte offset in array a
    add t3, a0, t2          # t3 = &a[b[i]]
    ld t3, 0(t3)            # Load a[b[i]] into t3

    # Calculate b[i] and subtract from a[b[i]]
    ld t2, 0(t2)            # Reload b[i], address is already in t2 from before
    sub t3, t3, t2          # t3 = a[b[i]] - b[i]

    # Add to total
    add a3, a3, t3          # total += a[b[i]] - b[i]

    addi t0, t0, 1          # i++
    j loop_start            # Jump back to start of loop

loop_end:
    mv a0, a3               # Move total to return register

    ld ra, 8(sp)            # Restore return address
    ld s0, 0(sp)            # Restore frame pointer
    addi sp, sp, 16         # Adjust stack back
    ret                      # Return to caller
```