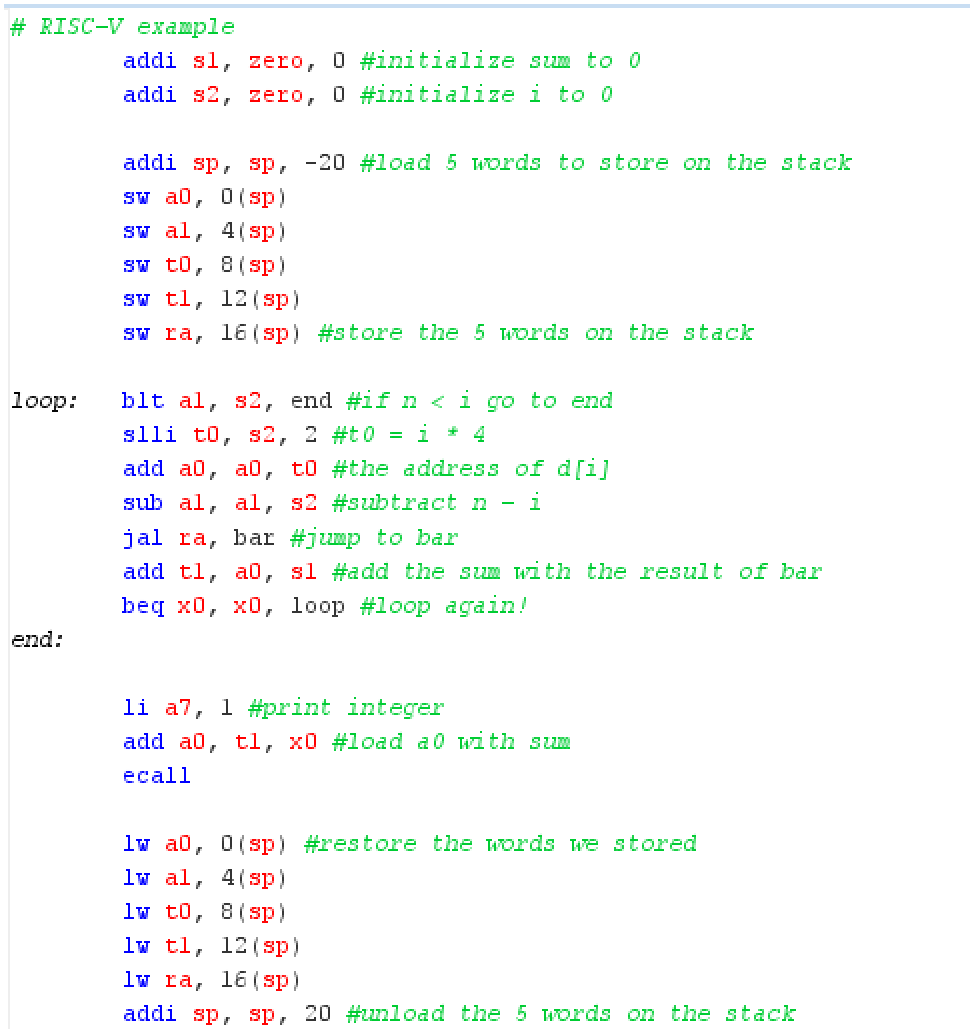
Archie Ross
2/17/2023

CSE 3666 Homework 3

1. **Translate Foo():**



```
# RISC-V example
        addi s1, zero, 0 #initialize sum to 0
        addi s2, zero, 0 #initialize i to 0

        addi sp, sp, -20 #load 5 words to store on the stack
        sw a0, 0(sp)
        sw a1, 4(sp)
        sw t0, 8(sp)
        sw t1, 12(sp)
        sw ra, 16(sp) #store the 5 words on the stack

loop:   blt a1, s2, end #if n < i go to end
        slli t0, s2, 2 #t0 = i * 4
        add a0, a0, t0 #the address of d[i]
        sub a1, a1, s2 #subtract n - i
        jal ra, bar #jump to bar
        add t1, a0, s1 #add the sum with the result of bar
        beq x0, x0, loop #loop again!
end:

        li a7, 1 #print integer
        add a0, t1, x0 #load a0 with sum
        ecall

        lw a0, 0(sp) #restore the words we stored
        lw a1, 4(sp)
        lw t0, 8(sp)
        lw t1, 12(sp)
        lw ra, 16(sp)
        addi sp, sp, 20 #unload the 5 words on the stack
```

In this screenshot I demonstrate the translation route I took for foo. I start by initializing i and sum, storing the 5 registers I used in the loop onto the stack, then running the loop. It directly correlates with the C code given. Once we call the bar and find the sum, we run the loop again until i is no longer smaller than n. Once i is larger than n, it will jump to the end and print the sum. Only once we've printed can we then restore all of the registers we stacked since we use a0 for the syscall. So then we restore our words, and add the load back on to the stack.

2. **Translate Msort():**

For this code, it was actually too big to include a single screenshot so I opted to also submit the file to HuskyCT. In the code, I start by storing a few registers including ra, and some save registers. I then have my exit branch for msort since msort is a recursive function. I then rewrote the pseudo code using my registers carefully. I then store my arguments before calling the function just in case the call alters them. Once those have been run, I load the arguments back and use them for the following function. Adjusting the registers accordingly as they are marked in my code and in the pseudo code. I repeated that process for the last 3 functions. Finally I unloaded all of the bits and restored the registers. And I called an exit system call because I was unsure what I was actually supposed to be returning. I wasn't sure if there was a specific type that copy may return in a0 that I need to know in order to print it in a system call. So I just ran the exit system call for fun.

3. **Encoding:**
    a. **BGE x10, x20, l100**
        i. Find out register numbers: x10, x20
        ii. SB-type
        iii. Immediate: 0001011
        iv. Opcode: 1100011
        v. Rd: 01000
        vi. Funct3: 101
        vii. Rs1: 01010
        viii. Rs2: 10100
        ix. Funct7: N/A
        x. Machine code in bits: 0001 0111 0100 0101 0101 0100 0110 0011
        xi. Machine code in hex: 0x17455263
    b. **BEQ x10, x0, l1**
        i. Find out register numbers: x10, x0
        ii. SB-type
        iii. Immediate: 1111110
        iv. Opcode: 1100011
        v. Rd: 11001
        vi. Funct3: 000
        vii. Rs1: 01010
        viii. Rs2: 00000
        ix. Funct7: N/A
        x. Machine code in bits: 1111 1100 0000 0101 0000 1100 1110 0011
        xi. Machine code in hex: 0xFC050CE3
    c. **JAL x0, l100**
        i. Find out register numbers: x0
        ii. UJ-type
        iii. Immediate: 11110110000111111111
        iv. Opcode: 1101111
        v. Rd: 00000
        vi. Funct3: N/A
        vii. Rs1: N/A
        viii. Rs2: N/A
        ix. Funct7: N/A
        x. Machine code in bits: 1111 0110 0001 1111 1111 0000 0110 1111
        xi. Machine code in hex: 0xF61FF06F

4. **Decoding:**
    a. **0xDB5A04E3**
        i. First convert to binary:
            1. 1101 1011 0101 1010 0000 0100 1110 0011
        ii. Identify type:
            1. 110 0011 opcode means it's a branch instruction
                a. SB-type
        iii. Identify Immediate:
            1. 1101101 + 01001
            2. 12|10:5 + 4:1|11
            3. 111011010100 + 0
            4. 1110110101000
            5. 0001001011000
            6. -600
        iv. Identify rs2 and rs1:
            1. Rs2: 10101 = x21
            2. Rs1: 10100 = x20
        v. Identify the Funct3:
            1. 000
                a. BEQ
        vi. Put it all together:
            1. BEQ   x20 x21 I-150
            2. Immediate: -600
            3. Target Address: 0x04003414
    b. **0xFA9FF0EF**
        i. First convert to binary:
            1. 1111 1010 1001 1111 1111 0000 1110 1111
        ii. Identify type:
            1. 110 1111 opcode
            2. UJ-Type
                a. JAL
        iii. Identify rd:
            1. Rd: 000 01 = x1
        iv. Identify Immediate:
            1. 20|10:1|11|19:12
            2. 1111 1111 1111 1101 0100
            3. 0000 0000 0000 0010 1100 + 0
            4. -88
        v. Put it all together:
            1. JAL   x1 I-22
            2. Immediate: -88
            3. Target Address: 0x04208830