Assignment 6, Part A

Adrian Lozada

March 24, 2023

1 Group

Group (Jiahui Dang)

2 Code Snippet

The NAND instruction is not part of the RISC-V instruction set because the same functionality can be implemented using existing instructions. Write a short assembly code snippet that has the following functionality: s3 = s4 NAND s5. Use as few instructions as possible.

```
and s3, s4, s5 # s3 = s4 & s5
xori s3, s3, -1 # s3 = ~s3
```

3 Code Snippet 2

Write RISC-V assembly code for placing the following immediate constants in register s7. Use a minimum number of instructions.

a)

```
addi s7, x0, 45 # s7 = 45
```

b)

```
addi s7, x0, -199 # s7 = -199
```

c)

```
lui s7, 0xFEDCC # Load the upper 20 bits of the immediate into s7
addi s7, s7, 0x8AB # Add the lower bits of the immediate to s7
```

d)

```
lui s7, 0xAABCD # Load the upper 20 bits of the immediate into s7
addi s7, s7, 0x325 # Add the lower bits of the immediate to s7
```

4 Code Snippet 3

Convert the following high-level code into RISC-V assembly language. Assume that the signed integer variables g and h are in registers t0 and t1, respectively

a)

```
if:
    bge t1, t0, else # t1 >= t0
    addi t0, t0, 9 # t0 = t0 + 9
    slli t2, t0, 2 # t2 = t0 * 4
    srai t3, t0, 1 # t3 = t0 / 2
    add t0, t2, t3 # t0 = t0 * 5
    j end
else:
    addi t1, t1, -5 # t1 = t1 - 5
    srai t1, t1, 3 # t1 = t1 / 8
```

b)

```
if2:
    blt t1, t0, else2 # t1 < t0
    slli t2, t1, 1 # t2 = t1 * 2
    add t1, t2, t1 # t1 = t1 * 3
    add t0, t1, t0 # t0 = t0 + t1
    andi t0, t0, 15 # t0 = t0 & 15
    j end2 # jump to end
else2:
    slli t0, t0, 1 # t0 = t0 * 2
    sub t0, t0, t1 # t0 = t0 - t1
    slli t2, t1, 3 # t2 = t1 * 8
    slli t3, t1, 2 # t3 = t1 * 4
    add t1, t2, t3 # t1 = t1 * 12
end2:
```

5 Code Snippet 4

Convert the following high-level code into RISC-V assembly language. Assume that the signed integer variables g and h are in registers t0 and t1, respectively. You can use other temporary registers like t2 and t3 if needed.

a)

```
        andi t2, t0, 15 # t2 = t0 % 16
        if:
            addi t3, x0, 9 # t3 = 9
            beq t2, t3, then # if t2 == t3
            addi t3, x0, 12 # t3 = 12
            beq t2, t3, then # if t2 == t3
            j else
        then:
            srai t1, t1, 1 # t1 = t1 / 2
            sub t1, t0, t1 # t1 = t0 - t1
            slli t2, t1, 1 # t2 = t1 * 2
            srai t3, t1, 1 # t3 = t1 / 2
            add t0, t2, t3 # t1 = t2 + t3
            add t0, t0, t1 # t0 = t0 + t1
            j end
        else:
            srai t2, t1, 1 # t2 = t1 / 2
            add t1, t1, t2 # t1 = t1 + t2
            add t1, t0, t1 # t1 = t1 + t0
            andi t0, t0, 7 # t0 = t0 % 8
        end:
```

b)

```
        andi t3, t0, 31 # t3 = t0 % 32
        if2:
            addi t2, x0, 6 # t2 = 6
            bge t2, t3, else2 # if t2 >= t3
            addi t2, x0, 17 # t2 = 17
            bge t2, t3, else2  # if t2 >= t3
            slli t2, t1, 2 # t2 = t1 * 4
            slli t3, t1, 1 # t3 = t1 * 2
            add t1, t2, t3 # t1 = t1 * 7
            add t1, t0, t1 # t1 = t1 + t0
            andi t0, t0, 3 # t0 = t0 % 4
            j end2
        else2:
            slli t2, t1, 3 # t2 = t1 * 8
            srai t3, t1, 1 # t3 = t1 / 2
```

```
        sub t1, t2, t3 # t1 = t2 - t3
        add t1, t0, t1 # t1 = t1 + t0
        slli t2, t1, 3 # t2 = t1 * 8
        sub t1, t2, t1 # t1 = t2 - t1
    end2:
```

6 Code Snippet 5

Comment on each snippet with what the snippet does. Assume that there is an array, int arr [6] = {3, 1, 4, 1, 5, 9}, which starts at memory address 0xBFFFFF00. You may assume each integer is stored in 4 bytes. Register a0 contains arr's address 0xBFFFFF00.

a)

```
# The code snippet takes the second and third element of the array 'arr' and adds them together.
# It then stores the result in the first element of the array 'arr'.

lw t0, 4(a0) # Load the value stored at the address 4(a0) into t0. This is the second element of the array.
lw t1, 8(a0) # Load the value stored at the address 8(a0) into t1. This is the third element of the array.
add t2, t0, t1 # Add t0 and t1 and store the result in t2
sw t2, 0(a0) # Store the value at register t2 at the address 0(a0). This is the first element of the array.
```

b)

```
# This code snippet implements a loop that iterates six times (the number of elements in the 'arr' array).
# During each iteration, it negates the value of the current element in the array.

    addi t0, a0, 0 # Initialize t0 to the start of the array
    addi t1, a0, 24 # Initialize t1 to the end of the array
loop: beq t0, t1, end # If t0 == t1, end the loop
    lw t2, 0(t0) # Load the value stored at the address stores in t0 into t2
    sub t2, x0, t2 # Negate t2
    sw t4, 0(t0) # Store the value in t4 at the address 0(t0)
    addi t0, t0, 4 # Increment t0 by 4
    j loop # Jump to the loop label
end:
```

7 Code Snippet 5

Write a RISC-V assembly snippet code to find the length of a string. Assume that the base address of string str is held in register a0. You may assume that each character is stored in 1 byte. You can use temporary registers if needed.

a)

```
        # Function:
        # t0 = pointer to array
        # t1 = current character
        # t2 = length of string
        add t0, x0, a0 # t0 = a0
        while:
            lbu t1, 0(t0) # t1 = *t0
            beq t1, x0, end # if t1 == 0, goto end
            addi t0, t0, 1 # t0++
            j while # goto while
        end:
            sub t2, t0, a0 # t2 = t0 - a0
```