Assignment 6, Part A

Adrian Lozada

March 22, 2023

1 Group

Independent

2 Code Snippet

The NOR 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 NOR s5. Use as few instructions as possible.

```
or s3, s4, s5 # s3 = s4 OR s5
xori s3, s3, -1 # s3 = s3 XOR -1
```

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, 59 # s7 = 59
```

b)

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

c)

```
lui s7, 0xDDCBE # Load the upper 20 bits of the
   immediate into s7
addi s7, s7, 0x289 # Add the lower 12 bits of the
   immediate to s7
```

d)

```
lui s7, 0x11236 # Load the upper 20 bits of the
   immediate into s7
lui s6, 0xBDF # Load the upper 20 bits of the
   immediate into s6
srli s5, s6, 12 # Shift s6 right by 12 bits
add s7, s7, s5 # Add s5 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. You can use other temporary registers like t2 and t3 if needed.

a)

```
    if:
        bge t1, t0, else # if (h >= g): else
        addi t0, t0, 7 # g = g + 7
        addi t2, zero, 1 # t2 = 1
        sll t3, t0, t2 # g * 2
        sra t0, t0, t2 # g / 2
        add t0, t0, t3 # g = g * 2 + g / 2
        jal x0, end
    else:
        addi t1, t1, -6 # g = g - 6
        srai t1, t1, 4 # g = g / 16
    end:
```

b)

```
    if2:
        blt t1, t0, else2 # if (g > h): else
        sub t0, t0, t1 # g = g - h
        srai t3, t0, 5 # g = g / 32
        slli t3, t3, 5 # g = g * g
        sub t0, t0, t3 # g = g - g * g
        jal a0, end2
    else2:
        slli t2, t1, 1 # h = h * 2
        add t1, t1, t2 # h = h * 3
        add t0, t1, t0 # g = g + h * 3
        slli t3, t0, 4 # h = g * 16
        slli t2, t0, 1 # h = g * 2
        sub t0, t3, t2 # g = g * 16 + g * 2
    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)

```
        srai t2, t0, 3 # divide by 8
        slli t2, t2, 3 # multiply by 8
        sub t3, t0, t2 # subtract
        addi t2, zero, 3 # set t2 to 3
        if:
            beq t3, t2, true # if t3 is equal to 3, go to true
            addi t2, zero, 5 # set t2 to 5
            beq t3, t2, true # if t3 is equal to 5, go to true
            jal x0, else
        true:
            slli t2, t1, 2 # multiply t1 by 4
            srai t3, t1, 1 # divide t2 by 4
            add t3, t3, t2 # add t3 and t2
            add t1, t1, t3 # add t1 and t3
            jal x0, end
        else:
            srai t3, t1, 4 # divide t1 by 16
            slli t3, t3, 4 # multiply t3 by 16
            sub t1, t1, t3 # subtract t3 from t0
        end:
```

b)

```
        srai t3, t0, 4 # divide t0 by 16
    slli t3, t3, 4 # multiply t3 by 16
    sub t3, t0, t3 # subtract t3 from t0
    if2:
        addi t2, zero, 4 # set t2 to 4
        beq t3, t2, else2 # if t3 is equal to 4, go to else
        blt t3, t2, else2 # if t3 is less than 4, go to else
        addi t2, zero, 12 # set t2 to 12
        bge t3, t2, else2 # if t3 is greater than or equal to
           12, go to else
        slli t3, t1, 2 # multiply t1 by 4
        add t1, t1, t3 # add t1 and t3
        add t1, t0, t1 # add t0 and t3
        srai t3, t1, 3 # divide t1 by 8
        slli t3, t3, 3 # multiply t3 by 8
        sub t1, t1, t3 # subtract t3 from t1
```

```
16          jal x0, end2
17      else2:
18          slli t2, t1, 3 # multiply t1 by 8
19          srai t1, t1, 1  # divide t1 by 2
20          add t1, t1, t2 # add t1 and t2
21          add t1, t1, t0 # add t1 and t0
22          slli t2, t1, 3 # multiply t1 by 8
23          add t1, t2, t1 # add t1 and t2
24      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 that each integer is stored in 4 bytes. Register a0 contains arr's address 0xBFFFFF00.

a)

```
# This code snippet loads the first and third elements of the 'arr' array,
  # adds them together, and stores the sum in the second element of the array.

  lw t0, 0(a0)    # Load the first element (3) of the 'arr' array into register t0.
  lw t1, 8(a0)    # Load the third element (4) of the 'arr' array into register t1.
  add t2, t0, t1  # Add the values in registers t0 and t1, storing the sum (7) in register t2.
  sw t2, 4(a0)    # Store the sum (value in t2) in the second element of the 'arr' array.
```

b)

```
      srai t3, t0, 4 # divide t0 by 16
  slli t3, t3, 4 # multiply t3 by 16
  sub t3, t0, t3 # subtract t3 from t0
  if2:
      addi t2, zero, 4 # set t2 to 4
      beq t3, t2, else2 # if t3 is equal to 4, go to else
      blt t3, t2, else2 # if t3 is less than 4, go to else
      addi t2, zero, 12 # set t2 to 12
      bge t3, t2, else2 # if t3 is greater than or equal to 12, go to else
      slli t3, t1, 2 # multiply t1 by 4
      add t1, t1, t3 # add t1 and t3
      add t1, t0, t1 # add t0 and t3
      srai t3, t1, 3 # divide t1 by 8
      slli t3, t3, 3 # multiply t3 by 8
      sub t1, t1, t3 # subtract t3 from t1
      jal x0, end2
  else2:
      slli t2, t1, 3 # multiply t1 by 8
      srai t1, t1, 1  # divide t1 by 2
      add t1, t1, t2 # add t1 and t2
      add t1, t1, t0 # add t1 and t0
      slli t2, t1, 3 # multiply t1 by 8
```

```
23          add t1, t2, t1 # add t1 and t2
24      end2:
```

7 Code Snippet 5

Write a RISC-V assembly snippet code to find the maximum and minimum elements in an array. Assume that the base address of array arr and the size of the array are held in register a0 and a1, respectively. You can use temporary registers if needed.

a)

```
    # The function:
    #  t0 = max
    #  t1 = min
    #  t2 = counter
    #  t3 = the length of the array
    #  t4 = value of array[i]

    lw t1, 0(a0) # Load the first value of the array into t1
    lw t0, 0(a0) # Load the first value of the array into t0
    add t2, zero, zero # Initialize the counter to 0
    lw t3, 0(a1) # Load the length of the array into t3
    lw t4, 0(a0) # Load the first value of the array into t4
    for:
        bge t2, t3, end # If the counter is gr. or eq to
           length go: end
        bge t1, t4, skip_min # Does not change the min value
        add t1, t4, zero # Change the min value
        skip_min:
        blt t4, t0, skip_max # Does not change the max value
        add t0, t4, zero # Change the max value
        skip_max:
        addi t2, t2, 1 # Increment the counter
        addi a0, a0, 4 # Increment the address of the array
        lw t4, 0(a0) # Load the next value of the array into t4
        jal x0, for # Go to the for loop
    end:
```