

# Assignment 6, Part A

Adrian Lozada

March 28, 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:  $s3 = s4 \text{ NAND } s5$ . Use as few instructions as possible.

```
1      and s3, s4, s5 # s3 = s4 & s5
2      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)

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

b)

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

c)

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

d)

```
1      lui s7, 0xAABCD # Load the upper 20 bits of the
                        immediate into s7
2      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)

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

b)

```
1      if2:
2          blt t1, t0, else2 # t1 < t0
3          slli t2, t1, 1 # t2 = t1 * 2
4          add t1, t2, t1 # t1 = t1 * 3
5          add t0, t1, t0 # t0 = t0 + t1
6          andi t0, t0, 15 # t0 = t0 & 15
7          j end2 # jump to end
8      else2:
9          slli t0, t0, 1 # t0 = t0 * 2
10         sub t0, t0, t1 # t0 = t0 - t1
11         slli t2, t1, 3 # t2 = t1 * 8
12         slli t3, t1, 2 # t3 = t1 * 4
13         add t1, t2, t3 # t1 = t1 * 12
14     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)

```
1      andi t2, t0, 15 # t2 = t0 % 16
2      if:
3          addi t3, x0, 9 # t3 = 9
4          beq t2, t3, then # if t2 == t3
5          andi t2, t2, 7 # t2 = t2 % 8
6          addi t3, x0, 12 # t3 = 12
7          beq t2, t3, then # if t2 == t3
8          j else
9      then:
10         srai t1, t1, 1 # t1 = t1 / 2
11         sub t1, t0, t1 # t1 = t0 - t1
12         slli t2, t1, 1 # t2 = t1 * 2
13         srai t3, t1, 1 # t3 = t1 / 2
14         add t1, t2, t3 # t1 = t2 + t3
15         j end
16     else:
17         srai t2, t1, 1 # t2 = t1 / 2
18         add t1, t1, t2 # t1 = t1 + t2
19         add t1, t0, t1 # t1 = t1 + t0
20         andi t0, t0, 7 # t0 = t0 % 8
21     end:
```

b)

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

```
16         sub t1, t2, t3 # t1 = t2 - t3
17         add t1, t0, t1 # t1 = t1 + t0
18         slli t2, t1, 3  # t2 = t1 * 8
19         sub t1, t2, t1  # t1 = t2 - t1
20     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 `0xBFFFFFF00`. You may assume each integer is stored in 4 bytes. Register `a0` contains `arr`'s address `0xBFFFFFF00`.

a)

```
1      # The code snippet takes the second (0xBFFFFFF04) and
      # third element (0xBFFFFFF08) of the array 'arr' and
      # adds them together.
2      # It then stores the result in the first element of
      # the array 'arr' (0xBFFFFFF00).
3
4      lw t0, 4(a0) # Load the value stored at the address
      # 4(a0) into t0. This is the second element of the
      # array.
5      lw t1, 8(a0) # Load the value stored at the address
      # 8(a0) into t1. This is the third element of the
      # array.
6      add t2, t0, t1 # Add t0 and t1 and store the result in
      # t2
7      sw t2, 0(a0) # Store the value at register t2 at the
      # address 0(a0). This is the first element of the
      # array.
```

b)

```
1      # This code snippet implements a loop that iterates
      # six times (the number of elements in the 'arr'
      # array).
2      # During each iteration, it negates the value of the
      # current element in the array.
3
4      addi t0, a0, 0 # Initialize t0 to the start of the
      # array
5      addi t1, a0, 24 # Initialize t1 to the end of the
      # array
6      loop: beq t0, t1, end # If t0 == t1, end the loop
7      lw t2, 0(t0) # Load the value stored at the
      # address stores in t0 into t2
8      sub t2, x0, t2 # Negate t2
9      sw t2, 0(t0) # Store the value in t2 at the
      # address 0(t0)
10     addi t0, t0, 4 # Increment t0 by 4
11     j loop # Jump to the loop label
12     end:
```

## 7 Code Snippet 6

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)

```
1      # Function:
2      # t0 = pointer to array
3      # t1 = current character
4      # t2 = length of string
5      add t0, x0, a0 # t0 = a0
6      while:
7          lbu t1, 0(t0) # t1 = *t0
8          beq t1, x0, end # if t1 == 0, goto end
9          addi t0, t0, 1 # t0++
10         j while # goto while
11     end:
12         sub t2, t0, a0 # t2 = t0 - a0
```