

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: $s3 = s4 \text{ NOR } s5$. Use as few instructions as possible.

```
1      or s3, s4, s5 # s3 = s4 OR s5
2      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)

```
1      addi s7, x0, 59 # s7 = 59
```

b)

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

c)

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

d)

```
1      lui s7, 0x11236 # Load the upper 20 bits of the
                        immediate into s7
2      lui s6, 0xBDF # Load the upper 20 bits of the
                        immediate into s6
3      srli s5, s6, 12 # Shift s6 right by 12 bits
4      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)

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

b)

```
1      if2:
2          blt t1, t0, else2 # if (g > h): else
3          sub t0, t0, t1 # g = g - h
4          srai t3, t0, 5 # g = g / 32
5          slli t3, t3, 5 # g = g * g
6          sub t0, t0, t3 # g = g - g * g
7          jal end2
8      else2:
9          slli t2, t1, 1 # h = h * 2
10         add t1, t1, t2 # h = h * 3
11         add t0, t1, t0 # g = g + h * 3
12         slli t3, t0, 4 # h = g * 16
13         slli t2, t0, 1 # h = g * 2
14         sub t0, t3, t2 # g = g * 16 + g * 2
15     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      srai t2, t0, 3 # divide by 8
2      slli t2, t2, 3 # multiply by 8
3      sub t3, t0, t2 # subtract
4      addi t2, zero, 3 # set t2 to 3
5      if:
6          beq t3, t2, true # if t3 is equal to 3, go to true
7          addi t2, zero, 5 # set t2 to 5
8          beq t3, t2, true # if t3 is equal to 5, go to true
9          jal else
10     true:
11         slli t2, t1, 2 # multiply t1 by 4
12         srai t3, t1, 1 # divide t2 by 4
13         add t3, t3, t2 # add t3 and t2
14         add t1, t1, t3 # add t1 and t3
15         jal end
16     else:
17         srai t3, t1, 4 # divide t1 by 16
18         slli t3, t3, 4 # multiply t3 by 16
19         sub t1, t1, t3 # subtract t3 from t0
20     end:
```

b)

```
1      srai t3, t0, 4 # divide t0 by 16
2      slli t3, t3, 4 # multiply t3 by 16
3      sub t3, t0, t3 # subtract t3 from t0
4      if2:
5          addi t2, zero, 4 # set t2 to 4
6          beq t3, t2, else2 # if t3 is equal to 4, go to else
7          blt t3, t2, else2 # if t3 is less than 4, go to else
8          addi t2, zero, 12 # set t2 to 12
9          bge t3, t2, else2 # if t3 is greater than or equal to
10         12, go to else
11         slli t3, t1, 2 # multiply t1 by 4
12         add t1, t1, t3 # add t1 and t3
13         add t1, t0, t1 # add t0 and t3
14         srai t3, t1, 3 # divide t1 by 8
15         slli t3, t3, 3 # multiply t3 by 8
16         sub t1, t1, t3 # subtract t3 from t1
```

```
16     jal 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 `0xBFFFFFF00`. You may assume that each integer is stored in 4 bytes. Register `a0` contains `arr`'s address `0xBFFFFFF00`.

a)

```
1 # This code snippet loads the first and third elements of the
  # 'arr' array,
2   # adds them together, and stores the sum in the second
  # element of the array.
3
4 lw t0, 0(a0)    # Load the first element (3) of the 'arr'
  # array into register t0.
5 lw t1, 8(a0)    # Load the third element (4) of the 'arr'
  # array into register t1.
6 add t2, t0, t1  # Add the values in registers t0 and t1,
  # storing the sum (7) in register t2.
7 sw t2, 4(a0)    # Store the sum (value in t2) in the
  # second element of the 'arr' array.
```

b)

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

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)

```
1      # The function:
2      #  t0 = max
3      #  t1 = min
4      #  t2 = counter
5      #  t3 = the length of the array
6      #  t4 = value of array[i]
7
8      lw t1, 0(a0) # Load the first value of the array into t1
9      lw t0, 0(a0) # Load the first value of the array into t0
10     add t2, zero, zero # Initialize the counter to 0
11     lw t3, 0(a1) # Load the length of the array into t3
12     lw t4, 0(a0) # Load the first value of the array into t4
13     for:
14         bge t2, t3, end # If the counter is gr. or eq to
15             length go: end
16         bge t1, t4, skip_min # Does not change the min value
17         add t1, t4, zero # Change the min value
18         skip_min:
19         blt t4, t0, skip_max # Does not change the max value
20         add t0, t4, zero # Change the max value
21         skip_max:
22         addi t2, t2, 1 # Increment the counter
23         addi a0, a0, 4 # Increment the address of the array
24         lw t4, 0(a0) # Load the next value of the array into t4
25         jal for # Go to the for loop
26     end:
```