CSE3666 — Lab 2

Mike Medved

February 1st, 2022

1 Prompt

In this lab, we write a program in RISC-V assembly language that prints 32 bits in a register.
The program reads a signed integer from the console and prints the 32 bits in the integer, twice.

Skeleton code in ‘lab2.s‘ already has the following steps. Study the code.

1. Read an integer using a system call and save it in ‘s1‘.
2. Use a system call to print ‘s1‘ in binary.

Implement the following steps with RISC-V instructions.

1. Use system call to print a newline character (ASCII value 10). Find the system call number in the document. Use system call to print a newline character (ASCII value 10). Find the system call number in the document. We can use n as the immediate in instructions.

2. Write a loop to print the bits in ‘s1‘, from bit 31 to bit 0. The printed bits should be the same as the ones printed by the system call. Think about the strategy/algorithm first. One method is provided in pseudocode at the bottom of this page.

3. Use system call to print a newline character.

Here are some example inputs/outputs of the program.

```
-1
11111111111111111111111111111111
11111111111111111111111111111111

3666
00000000000000000000111001010010
00000000000000000000111001010010

20220201
00000001001101001000100100101001
00000001001101001000100100101001

-3666
11111111111111111111000110101110
11111111111111111111000110101110
```

2 Deliverables

```
# CSE 3666 Lab 2

.globl main
.text

main:
    # use system call 5 to read integer
    addi    a7, x0, 5
    ecall
    addi    s1, a0, 0 # int in s1

    # use system call 35 to print a0 in binary
    # a0 has the integer we want to print
    addi    a7, x0, 35
    ecall

    # print newline
    addi    a7, x0, 11
    addi    a0, x0, 10
    ecall

    addi    a7, x0, 1   # set system call number to 1 (PrintInt)
    addi    t0, t0, 32  # i = 32 (traverse bits backwards)
    addi    t1, t1, 1   # keep 1 in t1 for the mask in loop
    j loop              # enter loop routine

    loop:
        addi t0, t0, -1 # decrement loop counter

        # (k & (1 << n)) >> n
        sll  t2, t1, t0 # t2 = (1 << i)
        and  t3, s1, t2 # t3 = num & (1 << i)
        srl  a0, t3, t0 # a0 = (num & (1 << i)) >> i)
        ecall           # print extracted bit

        # restart loop, or exit if done
        bne t0, x0, loop
        beq t0, x0, loop_exit

    loop_exit:
        # print newline
        addi    a7, x0, 11
        addi    a0, x0, 10
        ecall

    # exit program with exit code 0
    addi    a7, x0, 10
    ecall
```

3 Run Examples

```
43923
00000000000000001010101110010011
00000000000000001010101110010011

430940391
00011001101011111010000011100111
00011001101011111010000011100111

-5
11111111111111111111111111111011
11111111111111111111111111111011

-103012304
11111001110111000010100000110000
11111001110111000010100000110000
```

4 Limitations

This algorithm works for all 32-bit integers, $-2,147,483,647 \leq i \leq 2,147,483,647$, and only fails beyond that due to the range limitations of a 32-bit system. The actual algorithm itself, described by $(k \,\&\, (1 \ll n)) \gg n$, can work for any integer if these limitations are ignored, and that the loop counter is initialized to the proper amount of bits. Regardless, for the purposes of this lab, any valid integer input will work properly.