Lab 4. Arithmetic and Logical Instructions

Goals

After this laboratory exercise, you should know how to use arithmetic, logical and shift instructions. In addition, you should also understand overflow in arithmetic operation and how to detect it.

References

- RISC-V documents, textbook.
- The RISC-V Instruction Set Manual: riscv-spec-20191213.pdf

Preparation

Assignments at Home and at Lab

Home Assignment 1

The arithmatic was introduced in Laboratory Exercises 02, this assignment describes a special situation when adding two integers – the overflow.

Support: The sum of two 32-bit integers may not be representable in 32 bits. In this case, we say that an overflow has occurred. Overflow is possible only with operands of the same sign.
For two nonnegative (negative) operands, if the sum obtained is less (greater) than eitheir operand, overflow has occurred

The following program dectects overflow based on this rule. Two operands are stored in register **s1** and **s2**, the sum is stored in register **s3**. If overflow occur, **t0** register is set to 1 and clear to 0 in otherwise.

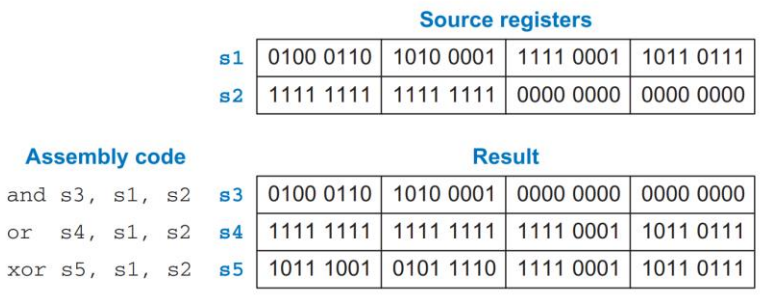
```
# Laboratory Exercise 4, Home Assignment 1
.text
   # TODO: Initialize s1 and s2 in different cases

   # Algorithm for determing orverflow condition
   li     t0, 0             # No overflow is default status
   add    s3, s1, s2        # s3 = s1 + s2
   xor    t1, s1, s2        # Test if s1 and s2 have the same sign
   blt    t1, zero, EXIT    # If not, exit
   slt    t2, s3, s1
   blt    s1, zero, NEGATIVE   # Test if s1 and s2 is negative?
   beq    t2, zero, EXIT       # s1 and s2 are positive
   # if s3 > s1 then the result is not overflow
   j      OVERFLOW
NEGATIVE:
   bne    t2, zero, EXIT    # s1 and s2 are negative
   # if s3 < s1 then the result is not overflow
```

```
OVERFLOW:
   li    t0, 1           # The result is overflow
EXIT:
```

Home Assignment 2

The basic logical operation includes **and**, **or**, **xor**, **not.**
These instructions operates with bits of source operands and write the result to destination operand.



Source registers

s1	0100 0110	1010 0001	1111 0001	1011 0111
s2	1111 1111	1111 1111	0000 0000	0000 0000

Assembly code		Result			
and s3, s1, s2	s3	0100 0110	1010 0001	0000 0000	0000 0000
or s4, s1, s2	s4	1111 1111	1111 1111	1111 0001	1011 0111
xor s5, s1, s2	s5	1011 1001	0101 1110	1111 0001	1011 0111

There are versions of immediate format of **and**, **or**, **xor** using with a source register and a 12-bit immediate value.

Some useful ways with logical operations:

i. **and** is used to extract some bits from a register or value
ii. **and** is used to clear some bits of a register or value
iii. **or** is used to set some bits for a register.

The following program demonstrates how to use logical instructions to extract information from one register. We can extract one bit or more according to the mask we use. Read this example carefully and explain each lines of:

```
# Laboratory Exercise 4, Home Assignment 2
.text
   li    s0, 0x12345678 # Test value
   andi  t0, s0, 0xff   # Extract LSB of s0
   andi  t1, s0, 0x0400 # Extract bit 10 of s0
```

Home Assignment 3

The next logical instructions in this assignment are shift instructions including `sll` (shift left logical), `srl` (shift right logical), and `sra` (shift right arithmetic).

		Source register			
	s5	1111 1111	0001 1100	0001 0000	1110 0111

Assembly code		**Result**			
`slli t0, s5, 7`	**t0**	1000 1110	0000 1000	0111 0011	1000 0000
`srli s1, s5, 17`	**s1**	0000 0000	0000 0000	0111 1111	1000 1110
`srai t2, s5, 3`	**t2**	1111 1111	1110 0011	1000 0010	0001 1100

RISC-V supports the immediate format of the shift instructions (`slli`, `srli`, and `srai`), with 5-bit constant.
Note that, shifting left (right) n-bit is equivalent to multiply (divide) by 2^n.
This example show how the shift operations used to implement other instructions, such as multiply by a small power of 2

```
# Laboratory Exercise 3, Home Assignment 3
.text
   li    s0, 1        # s0 = 1
   sll   s1, s0, 2    # s1 = s0 * 4
```

Assignment 1

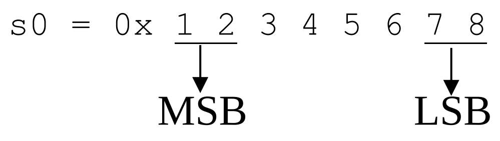
Create a new project to implement the Home Assigment 1. Compile and upload to simulator. Initialize two operands (register s1 and s2), run this program step by step, observe memory and registers value.

Assignment 2

Write a program to do the following tasks:

- Extract MSB of s0
- Clear LSB of s0
- Set LSB of s0 (bits 7 to 0 are set to 1)
- Clear s0 (s0=0, must use logical instructions)

MSB: Most Significant Byte
LSB: Least Significant Byte



Assignment 3

Pseudo instructions in RISC-V are not-directly-run-on-RISC-V-processor instructions which need to be converted to real-instructions of RISC-V. Re-write the following pseudo instructions using real-instructions understood by RISC-V processors:

a. `abs    s0, s1`
```
     s0 = abs($s1)
```

b. `move  s0, s1`
```
     s0 = s1
```
c. `not   s0`
```
     s0 = bit_invert(s0)
```
d. `ble   s1, s2, label`
```
     if (s1 <= s2)
          j label
```

Assignment 4

To dectect overflow in additional operation, we also use other rule than the one in Assignment 1. This rule is: when add two operands that have the same sign, overflow will occur if the sum doesn't have the same sign with either operands. You need to use this rule to write another overflow detection program.

Assignment 5

Write a program that implement multiply by a small power of 2. (2, 4, 8, 16, etc for example).

Conclusions

What is benefit of using shift instructions for implementing multiplication comparing with multiply instructions (RV32M: RISC-V 32-bit Multiplier/Divider) ?