Logical and Bitwise Operations

Caiwen Ding

Department of Computer Science and Engineering

University of Connecticut

CSE3666: Introduction to Computer Architecture

Outline

- Compare unsigned numbers
- Load 32-bit constants
- Bitwise and logical operations
    - And related RISC-V instructions

NOT, AND, OR, XOR

Shift left, shift right logical, shift right arithmetic

- Application of these operations

Reading: Sections 2.6. Skip instruction encoding.

Reminder: Reference card

Review Question

Which register is larger? *Value in* Does the following instruction jump to L?

```
if (t0 < t1) goto L
```

signed } ?
unsigned }

t0	1111 1111 1111 1111 1111 1111 1111 1111

t1	0000 0000 0000 0000 0000 0000 0000 0001

Question

Which register is larger? Does the following instruction jump to L?

```
if (t0 < t1) goto L
```

blt

t0	1111 1111 1111 1111 1111 1111 1111 1111

t1	0000 0000 0000 0000 0000 0000 0000 0001

signed: t0 < t1 because -1 < 1
unsigned: t0 > t1 because $(2^{32} - 1) > 1$

4294967295

Branches, with unsigned comparison

- Conditional branches
  - If a condition is true, go to the instruction indicated by the label
  - Otherwise, continue sequentially

do we need to separate signed & unsigned ? ✗

```
beq    rs1, rs2, L1 # if (rs1 == rs2) goto L1
bne    rs1, rs2, L2 # if (rs1 != rs2) goto L2

# compare signed numbers      *signed*
blt    rs1, rs2, L3 # if (rs1 <  rs2) goto L3
bge    rs1, rs2, L4 # if (rs1 >= rs2) goto L4

# compare unsigned numbers    *unsigned*
bltu   rs1, rs2, L  # if (rs1 <  rs2) goto L
bgeu   rs1, rs2, L  # if (rs1 >= rs2) goto L
```

↳ unsigned

Example

```
# s1 is the number of elements in an array
# check if index t0 is in range [0, s1)
# both s1 and t0 are signed

if (t0 < 0) || (t0 >= s1) goto L_error
```

HLL → high-level language

```
  bgeu  t0, s1, L_error
```

unsigned   unsigned

```
# how would you check if t0 is in [s2, s1)?
```

check yourself

Load 32-bit Constants into a Register

- We are good at 12-bit immediate most of the time, but sometimes need larger numbers
- How do we load a 32-bit constant in a register?

Example:

0x12345678

31	...						0
0001	0010	0011	0100	0101	0110	0111	1000

Range

range [-2048, 2047]

range [-2048, 2047]

immd

```
addi    a0, x0, 0xFF  →12
addi    a1, x0, 255
```

0x00400000	0x0ff00513	addi x10,x0,0x000000ff	16:		addi	a0, x0, 0xFF
0x00400004	0x0ff00593	addi x11,x0,0x000000ff	17:		addi	a1, x0, 255

a0		10	0x000000ff
a1		11	0x000000ff

LUI

(handwritten: 32 Regs → 5 bit)
(handwritten: (20-bit))
(handwritten: Data Value / Instruction } 32-bit)

(handwritten: Load Upper Immd)
(handwritten: why 20 bit?)
(handwritten: LUI rd, immd)
(handwritten: 20 (bits))
(handwritten: 7bit 5 32bit)

- LUI allows 20-bit immediate
  - Assembler supports %hi(C) to get the higher 20 bits of C
- The 20 bits are placed into bits 12 to bits 31
  - Lower 12 bits are cleared

(handwritten: immd)

31 ...	12	11	0
xxxx xxxx xxxx xxxx xxxx		0000 0000 0000	

20 bits from immd
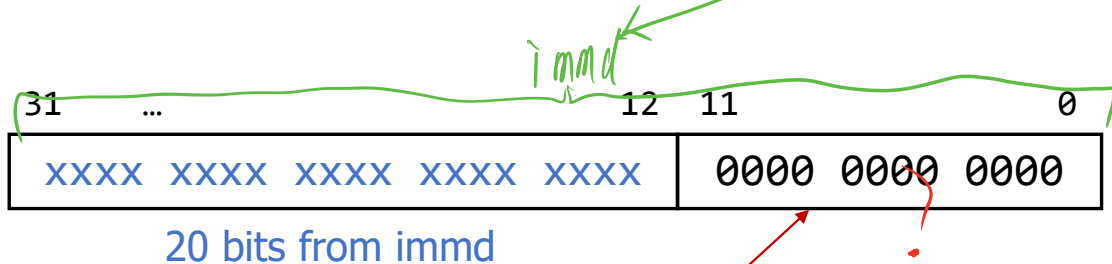
How do we set the lower 12 bits to other values?

Example: load large constants

- Load 0x12345678 into register s0

```
lui   s0, 0x12345
addi  s0, s0, 0x678
```
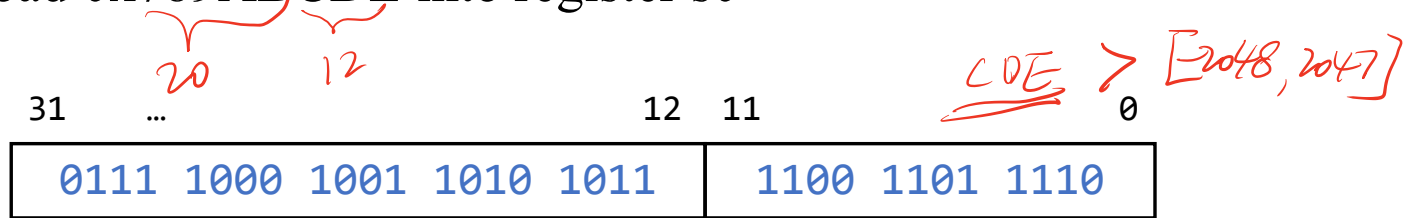
(handwritten annotations: 20 bits → 32 bits, 12 bits, 32-bit, li (load immd) x exam, HW, ✓ Lab)

```
0x00400008  0x12345537 lui x10,0x00012345   16:    li    a0, 0x12345678
0x0040000c  0xe7850513 addi x10,x10,0x000678
```

31 … 12	11 0
0001 0010 0011 0100 0101	0110 0111 1000

Example: load large constants

- Load 0x789ABCDE into register s0

Example: load large constants

- Load 0x789ABCDE into register s0

```
lui x10,0x000789ac    16:    lui    a0, 0x789AC
addi x10,x10,0xffff... 17:   addi   a0, a0, 0xFFFFFCDE
```

```
a0                                    10        0x789abcde
```
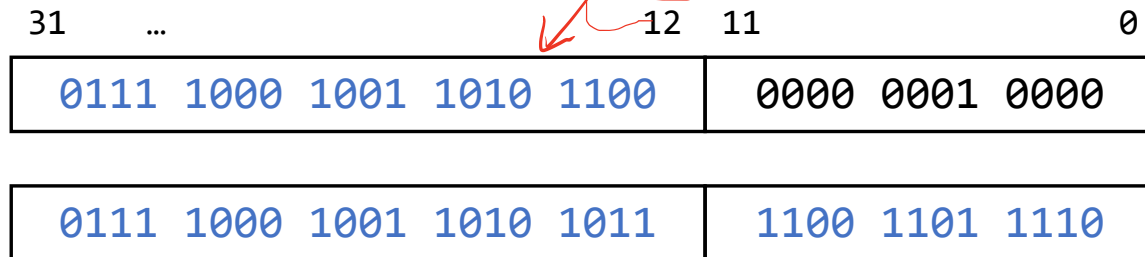
```
lui    s0, 0x789AC
addi   s0, s0, 0xFFFFFCDE    # sign extended !
```

31 ...	12	11	0

0111 1000 1001 1010 1100	0000 0001 0000

0111 1000 1001 1010 1011	1100 1101 1110

-1 is added to upper 20 bits, because lower 12 bits are sign extended

```
addi    a0, a0, 0xCDE
```

Error in P:\Lectures\cse3666_demo\01-hello.s line 17 column 15: "0xCDE": operand is out of range

Bitwise logical operations: NOT, AND, OR, and XOR

NOT ~

Truth Table

X	NOT X
0	1
1	0

① input
② output
③ 'function'

Truth Table

XOR ^

X	Y	X XOR Y
0	Same	0	0
0	1	1
1	Not Same	0	1
1	1	0

AND &

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

OR |

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

12

Examples: 8-bit bitwise logical operations

we treat bits seperately

A	1001 1011
B	1100 1101
A AND B	1000 1001

A	1001 0011
B	1101 1001
A OR B	1101 1011

A	1101 1011
B	1001 1111
A XOR B	0100 0100

A	1001 1011
NOT A	0110 0100

Example: 8-bit shift operations

Shift left	1001 1011
By 3 (<< 3)	1101 1000

Shift right logical	1001 1011
By 4 (>> 4)	0000 1001

Shift right arith.	1001 1011
By 4 (>> 4)	1111 1001

There are two versions of shift right.
The sign bit is padded in from the left for shift right arithmetic

RISC-V Support for Logical Operations

Operation	C/Python	RISC-V
Shift left	<<	sll, slli
Shift right logic	>>	srl, srli
Shift right arith.	>>	sra, srai
Bitwise AND	&	and, andi
Bitwise OR	|	or, ori
Bitwise NOT	~	xori
XOR	^	xor, xori

*i instructions take an immediate as the second operand
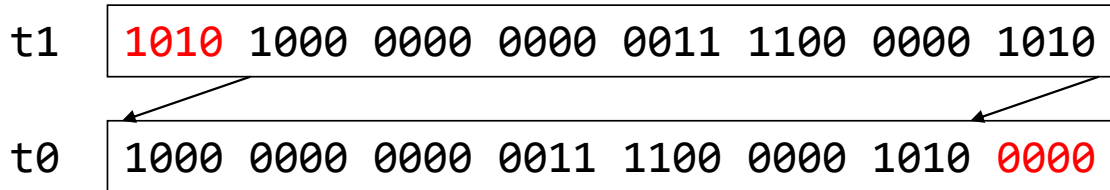Immediates are 12-bit long and sign extended

SLLI and SLL Operations

```
slli t0, t1, 4
sll  t0, t1, t2      # assume t2 is 4
```

Shift the bits in t1 left by 4 positions

The highest 4 bits in t1 are lost. 0 is shifted in from the right

t1 | 1010 1000 0000 0000 0011 1100 0000 1010

t0 | 1000 0000 0000 0011 1100 0000 1010 0000

SRLI and SRAI Operations

```
srli t2, t1, 4       # srl takes registers
srai t3, t1, 4       # sra takes registers
```

- Shift the bits right by 4 positions
  – SRLI pads with 0 and SRAI pads with the sign bit (not always 1!)

t1	1010 1000 0000 0000 0011 1100 0000 1010

t2	0000 1010 1000 0000 0000 0011 1100 0000

t3	1111 1010 1000 0000 0000 0011 1100 0000

AND and ANDI Operations

```
and    t0, t1, t2
andi   t0, t1, 0x5CC
```

The 12-bit immediate in ANDI is sign extended

```
       t1  0000 0000 0000 0000 0011 0100 0101 0111

t2 or immd  0000 0000 0000 0000 0000 0101 1100 1100

       t0  0000 0000 0000 0000 0000 0100 0100 0100
```

What do we see if we consider immd/t2 is a mask?

OR and ORI Operations

```
or    t0, t1, t2
ori   t0, t1, 0xFFFFFDC0
```

The 12-bit immediate is sign extended

t1	1000 0100 0000 0000 0101 0000 1101 1010
t2 or immd	1111 1111 1111 1111 1111 1101 1100 0000
t0	1111 1111 1111 1111 1111 1101 1101 1010

What do we see if we consider immd/t2 is a mask?

XOR and XORI Operations

```
xor    t0, t1, t2
xori   t0, t1, 0x5CF
```

The 12-bit immediate is sign extended

t1	0101 1110 1111 1111 1100 1100 1001 1110
t2 or immd	0000 0000 0000 0000 0000 0101 1100 1111
t0	0101 1110 1111 1111 1100 1001 0101 0001

What do we see if we consider immd/t2 is a mask?

NOT Operation

- Invert bits in a doubleword
  - Change 0 to 1, and 1 to 0
- RISC-V does not have NOT. NOT is done with an XOR
  - NOT is a pseudoinstruction

```
not t0, t1    # xori t0, t1, -1
```

t1	1111 0000 0000 0000 0011 1100 0000 0000

immd	1111 1111 1111 1111 1111 1111 1111 1111

t0	0000 1111 1111 1111 1100 0011 1111 1111

Question

What are the bits in t0 after the following instruction?

```
ori   t0, t1, -1
```

A. All bits in t0 are 1
B. 32 bits from t1
C. Higher 20 bits are from t1. Lower 12 bits are set to 0
D. Higher 20 bits are from t1. Lower 12 bits are set to 1
E. None of the above

Question

Write RISC-V instructions to perform the following operations.
How many instructions do you need for each multiplication?

```
# s1 = s0 * 4


# s1 = s0 * 128


# s1 = s0 * 9


# s1 = s0 * 7
```

Question

Write RISC-V instructions for the following operation.

```
s1 / 4                    # integer division
```

If v is not divisible by 4, the result is rounding towards negative infinity
If v is negative, it may not be what you want

Question

Suppose v = 0b 0100 1100. What is the binary representation of the following values? How do you use one RISC-V instruction to compute each of them?

```
v % 4
v % 8
v % 16
```

Question

Write RISC-V instructions for the following operations.

```
if s0 % 4 == 0 goto L4
```

Question

What instruction should be placed in the blank?

```
if s0 is even goto L2

    andi    t0, s0, 1
    ____    t0, x0, L2
```

A. BEQ

B. BNE

C. BLT

D. BGE

E. Need to change the registers in the second instruction

Logical operators in high-level languages

- How do we do logical operators AND and OR in C/Python?

```
// Python: and, or
// C: &&, ||

if (cond1 && cond2) {
        // if_branch
} else {
        // else_branch
}


Example:
if ((p != NULL) && (p[0] < 0)) { …
```

Short-circuit evaluation!

If cond1 is not true,
cond2 is not evaluated.

Logical Operators

```
if (cond1 and cond2) then
        if_branch
Else
        else_branch
```

Pseudocode
```
if ! cond1 goto Else
if ! cond2 goto Else
        if_branch
        goto EndIf
Else:
        else_branch
EndIf:
```

```
if (cond1 or cond2) then
        if_branch
Else
        else_branch
```

Pseudocode
```
if cond1 goto If
if ! cond2 goto else
If:
        if_branch
        goto EndIf
Else:
        else_branch
EndIf:
```