Logical and Bitwise Operations

Z. Jerry Shi
Department of Computer Science and Engineering
University of Connecticut

CSE3666: Introduction to Computer Architecture

Outline

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

NOT, AND, OR, XOR

Shift left, shift right logical, shift right arithmetic

- Application of these operations

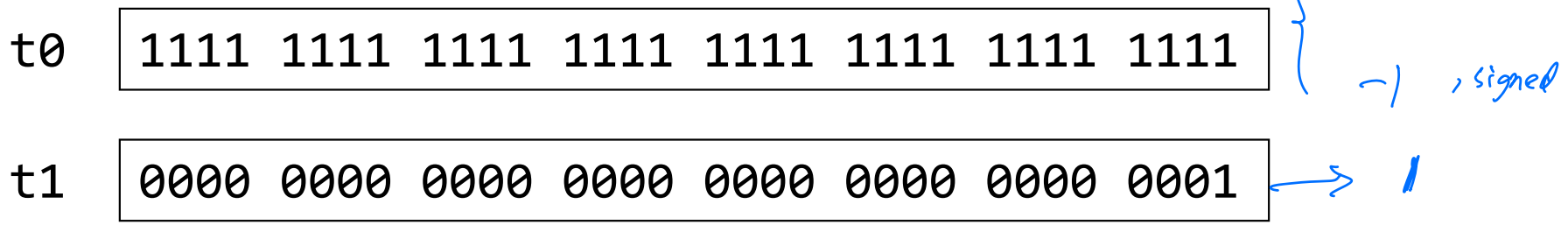
Reading: Sections 2.6. Skip instruction encoding.

Reminder: Reference card

Review Question

Which register is larger? Is the condition true?

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



Question

Which register is larger?

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

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

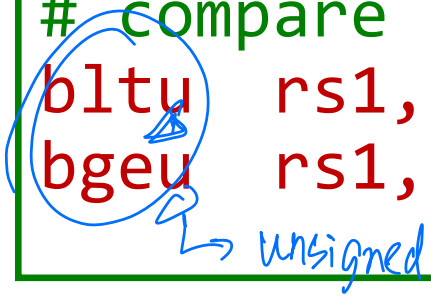
Branches, with unsigned comparison

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

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

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

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



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

# translate directly to RISC-V
   blt   t0, x0, L_error
   bge   t0, s1, L_error

# a trick
   bgeu  t0, s1, L_error
```

Logical operations: NOT, AND, OR, and XOR

NOT ~

X	NOT X
0	1
1	0

Truth Table

XOR ^

X	Y	X XOR Y
0	0	0
0	1	1
1	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

Examples: 8-bit bitwise logical operations

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

NOT. X1, X2 X real inst

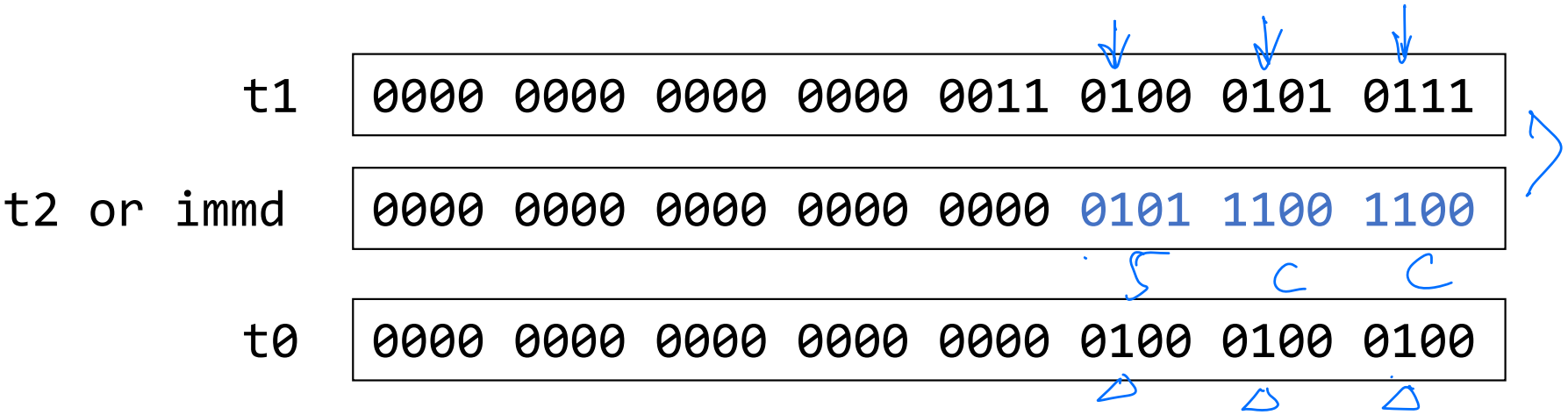
*i instructions take an immediate as the second operand

Immediates are 12-bit long and sign extended

AND and ANDI Operations

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

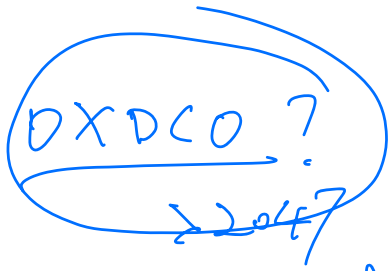
The 12-bit immediate in ANDI is sign extended



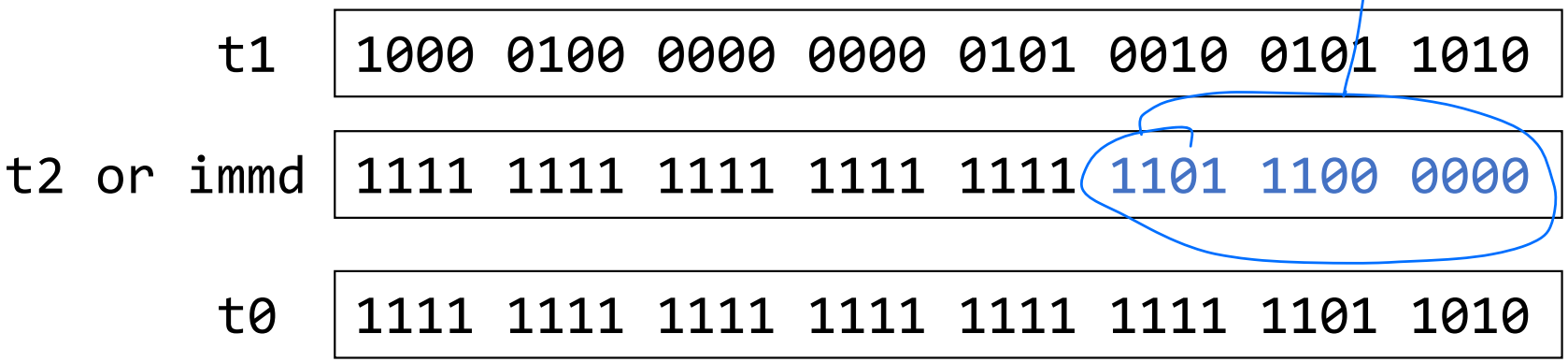
OR and ORI Operations

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

The 12-bit immediate is sign extended



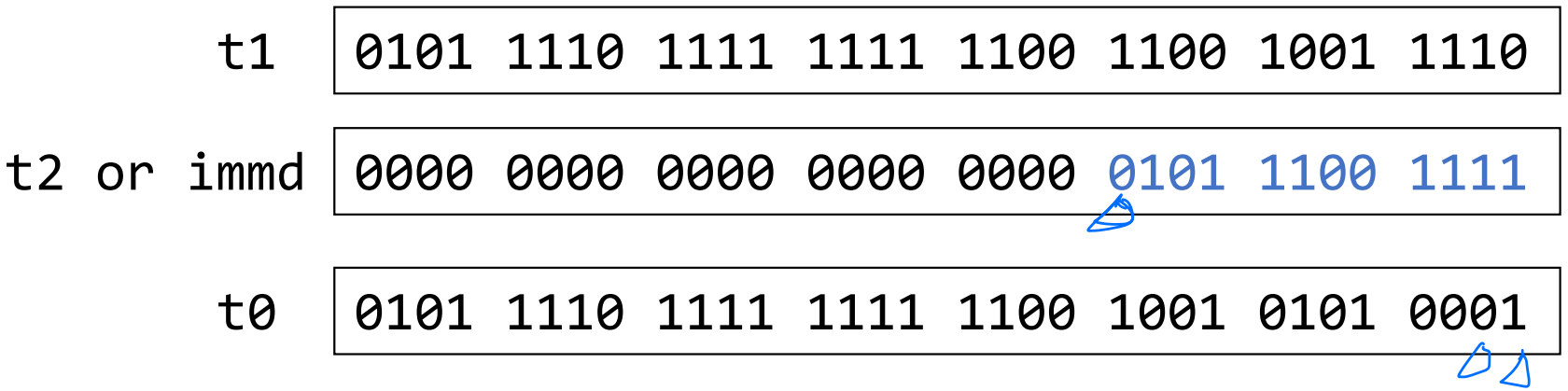
Range : [-2048, 2047]



XOR and XORI Operations

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

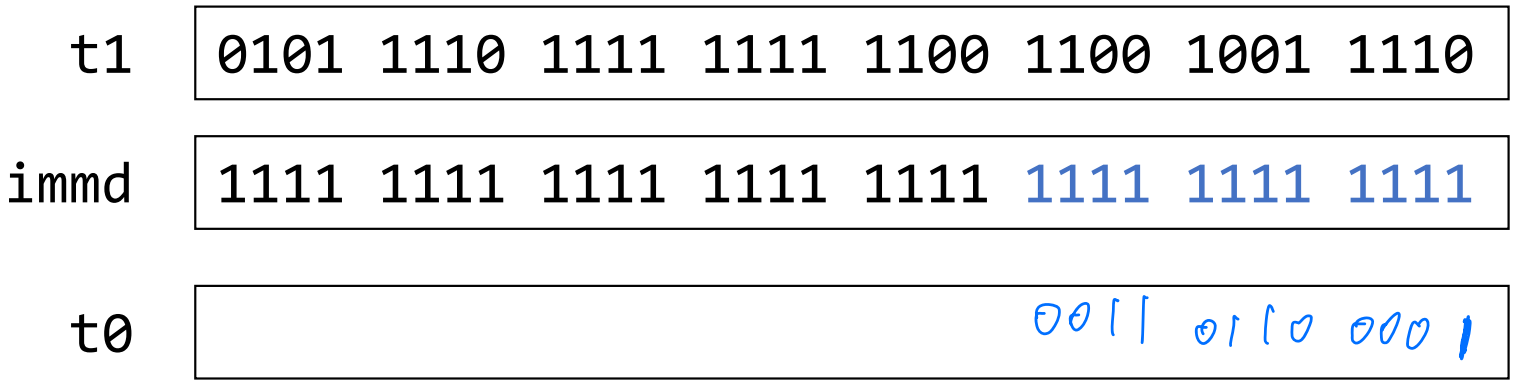
The 12-bit immediate is sign extended



XORI Example

```
xori  t0, t1, -1
```

The 12-bit immediate is sign extended



What is this operation?

NOT Operation

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

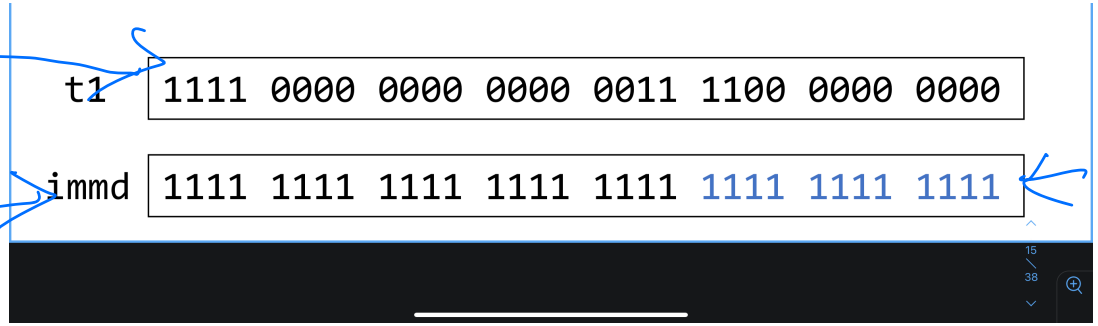
```
not t0, t1    # xori t0, t1, -1
```

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
```



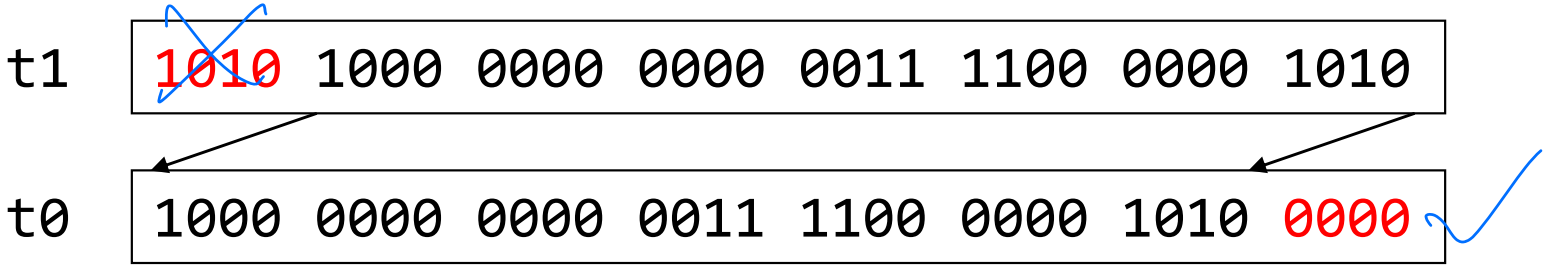
A. All bits in t0 are 1
B. All bits in t0 are 0
C. 32 bits from t1
D. Higher 20 bits are from t1. Lower 12 bits are set to 0
E. Higher 20 bits are from t1. Lower 12 bits are set to 1

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

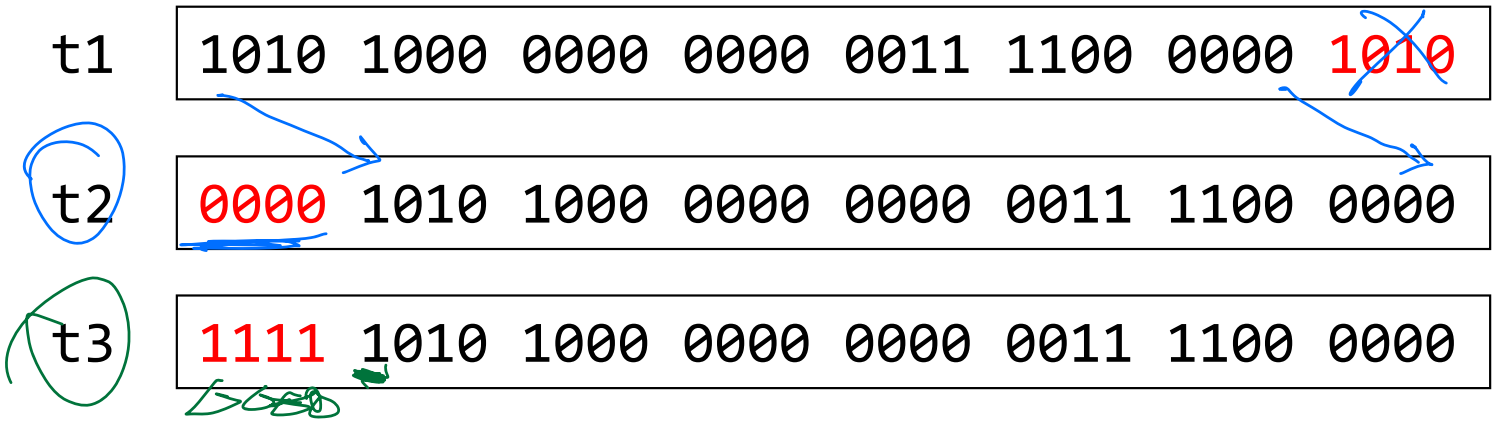
The highest 4 bits in t1 are lost. 0 is shifted in from the right



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!)



Question

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

```
# s1 = s0 * 4
```

slli s1, s0, 2

```
# s1 = s0 * 128
```

slli s1, s0, 7

```
# s1 = s0 * 9
```

slli s1, s0, 3 # s1 = s0*8
add s1, s1, s0 # s1 = s0*8 + s0 = s0*9

```
# s1 = s0 * 7
```

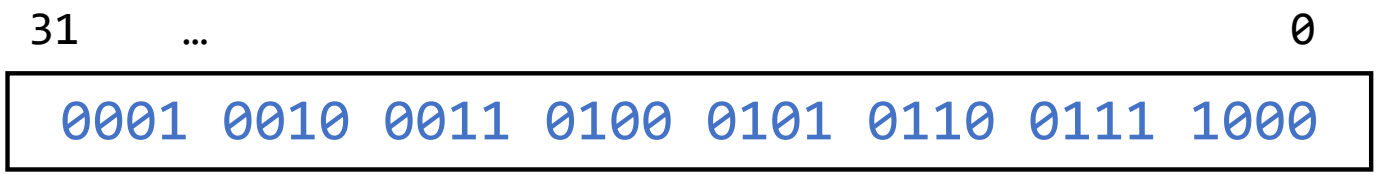
slli s1, s0, 3
sub s1, s1, s0

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:

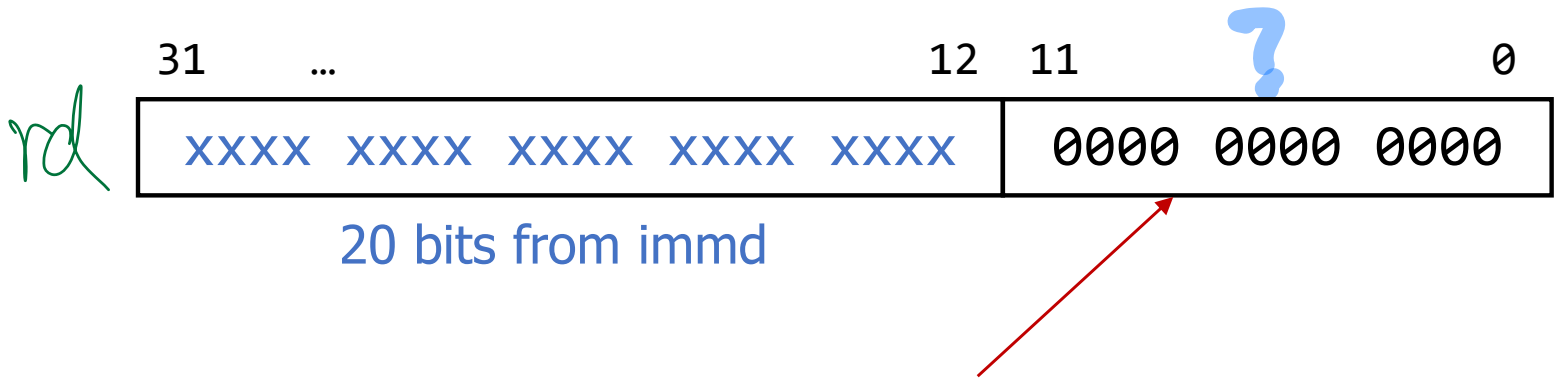
0x12345678 $\neq (12345678)_{10}$



LUI (Load Upper Immd)

```
LUI rd, immd
```

- 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



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

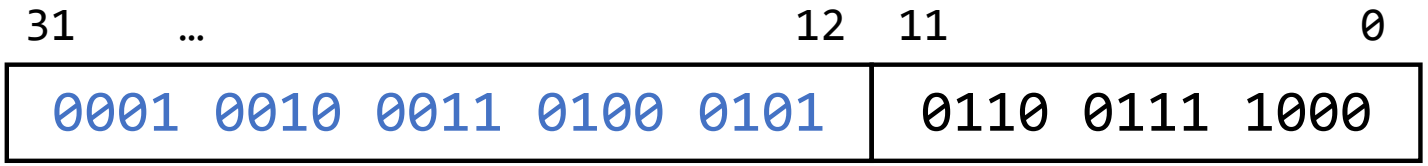
Example: load large constants

- Load 0x12345678 into register s0

```
lui   s0, 0x12345
addi  s0, s0, 0x678
```

hi 20 → s0 = 0001 0010 0011 0100 0101 0000 0000 0000 (low 12)

low 12 → s0 = 0001 0010 0011 0100 0101 0110 0111 1000 (low 12) ✓



Example: load large constants

- Load 0x789ABCDE into register s0

3294 > [-2048, 2047]

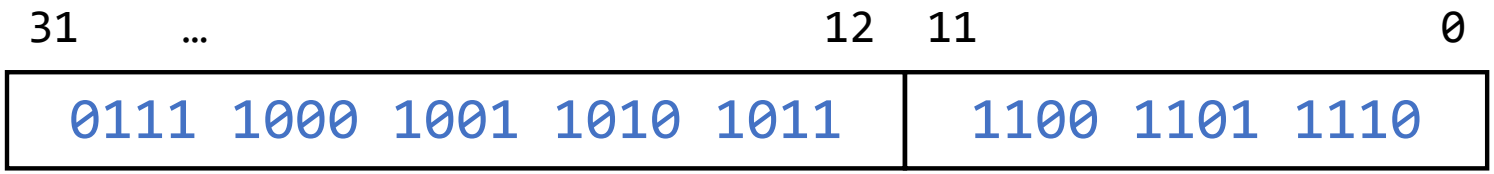
can not use 0xCDE for addi

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

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

```
lui x10,0x000789ab      16:     lui     a0, 0x789AB
addi x10,x10,0xffff...  17:     addi    a0, a0, 0xFFFFFCDE
```

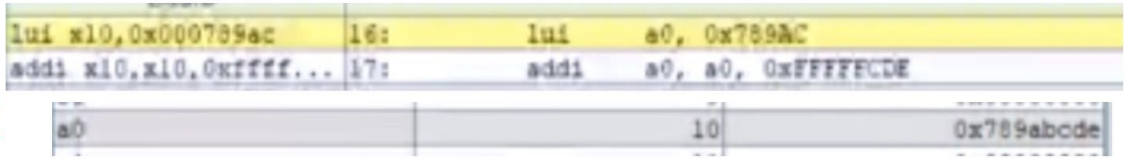
```
a0                              10              0x789aacde
```

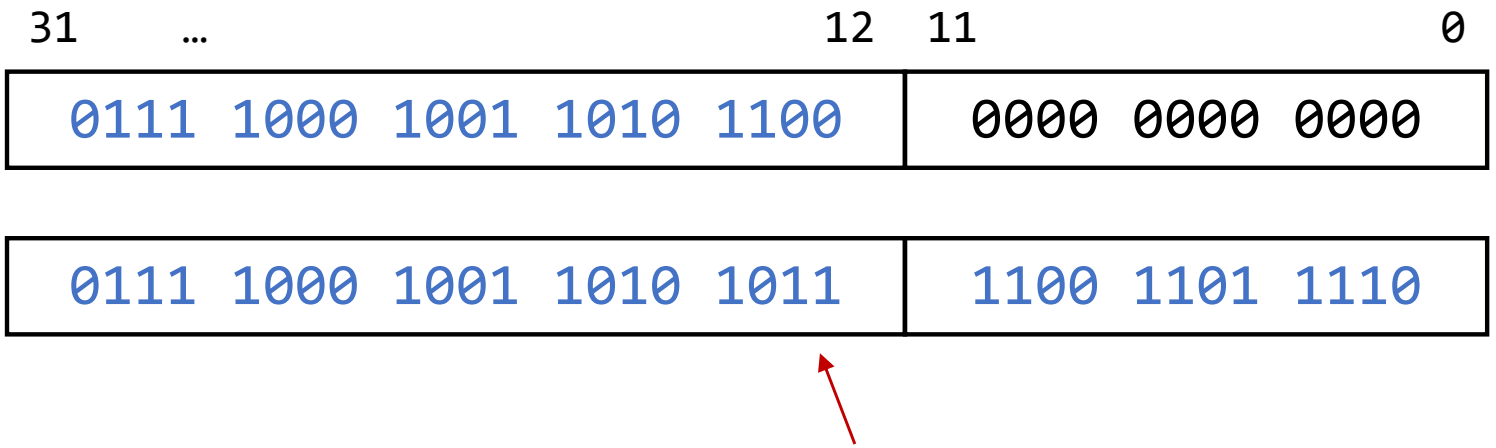


Example: load large constants

- Load 0x789ABCDE into register s0

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





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

Study the remaining slides yourself

Logical operators

- Logical operators in Python: `and, or, not`
- Logical operators in C: `&&, ||, !`

In logical expressions, non-zero values are True

What is the result of the following expressions?

```
0x10 and 0x01

0x10 & 0x01

# Try the following in Python: ~1 vs not 1
```

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
}
```

Short-circuit evaluation!

If cond1 is not true,
cond2 is not evaluated.

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

if obj is not None and obj.v :          # Python
```

Logical Operators in If Statements

```
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:
```

Q&A

- Should I use instructions for signed or unsigned values? How do I know?

- C compiler knows the data type of variables
  - For example: int vs unsigned int
- When writing RISC-V code, we have to keep track of the data type ourselves

- If sign does NOT matter in an operation, there is only one instruction, e.g., add/sub/and/beq
- If sign does matter, there are two instructions
  - For example, blt vs bltu, sra vs srl

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 0000 0000 0000 0000 1111 1111 0100 1101.

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

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

Question

Write RISC-V instructions for the following operations.

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

Exercise

- Write RISC-V instructions to perform

```
s2 = isdigit(s1)        # '0' <= s1 <= '9'
```

Question

Write RISC-V code to flip bits 0, 1, and 4 of register s0 and keep other bits unchanged

s0	1111 1010 0000 0000 0011 1100 0110 1001
s0'	1111 1010 0000 0000 0011 1100 0111 1010

Question

Write RISC-V code to set bit 4 to bit 10 of register s0 to 1 and keep other bits unchanged

s0	1111 1010 0000 0000 0011 1100 0110 1001
s0'	1111 1010 0000 0000 0011 1111 1111 1010

Question

Write RISC-V code to clear lower eight bits of register s0 and keep other bits unchanged

s0	1111 1010 0000 0000 0011 1100 0110 1001
s0'	1111 1010 0000 0000 0011 1100 0000 0000

Question

What is the result of the following operation?

Show your answer as four hex digits.

```
0x0FBA ^ 0xFF98
```

```
0x0F0F & 0xABCD
```

```
0x0F0F | 0x3666
```

vert bits in a word

Change 0 to 1, and 1 to 0

SC-V does not have NOT. NOT is done with an XOR

NOT is a pseudoinstruction supported by assembler

```
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

vert bits in a word

Change 0 to 1, and 1 to 0

SC-V does not have NOT. NOT is done with an XOR

NOT is a pseudoinstruction supported by assembler

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