

Bitwise Operations

- Bitwise operations in high-level languages are applied to integers
- Java has three primary sizes for signed integers
 - short (16 bit chunks)
 - int (32 bit chunks)
 - long (64 bit chunks)
- Two types of Bitwise Operations
 - Boolean based operations
 - Shift-based operations

- Boolean-based Operations:

- Complement: `s1 = ~ t1;` `nor $s1, $t1, $zero # s1 = ~ (t1 | 0)`
- And: `s1 = t1 & t2;` `and $s1, $t1, $t2`
- Or: `s1 = t1 | t2;` `or $s1, $t1, $t2`
- Xor: `s1 = t1 ^ t2;` `xor $s1, $t1, $t2`

- Shift-based Operations:

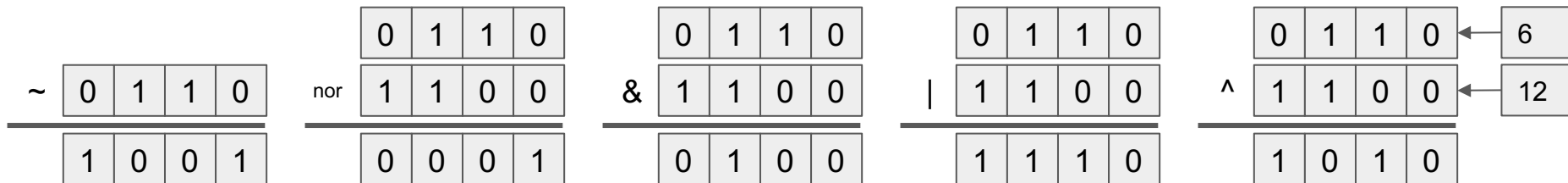
- ~~◦ Un/Signed Left Shift~~ `s1 = t1 << 2;` `sll $s1, $t1, 2 # Shift Left Logical`
- Unsigned Right Shift `s1 = t1 >>> 2;` `srl $s1, $t1, 2 # Shift Right Logical`
- Signed Right Shift `s1 = t1 >> 2;` `sra $s1, $t1, 2 # Shift Right Arithmetic`
- ~~◦ Unsigned Left Shift~~ ~~`s1 = t1 <<< t2;`~~

Boolean-based Bitwise Operations

- Let's assume 4-bit chunks:

- Complement: $s1 = \sim t1$ `nor $s1, $t1, $zero`
- And: $s1 = t1 \& t2$ `and $s1, $t1, $t2`
- Or: $s1 = t1 | t2$ `or $s1, $t1, $t2`
- Xor: $s1 = t1 \wedge t2$ `xor $s1, $t1, $t2`

A	B	nor	&		^
0	0	1	0	0	0
0	1	0	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0



Shift-based Operations

Foreshadow:

- Integers are encoded in 2's complement
- In such numbers, the MSb is represents the sign
- 1 -> a negative number

- Java and MIPS supported:

- Shift Left Logical
- Shift Right Logical
- Shift Right Arithmetic
- Arithmetic Shifting a value in a register
 - `sllv`, `srlv`, `srav`

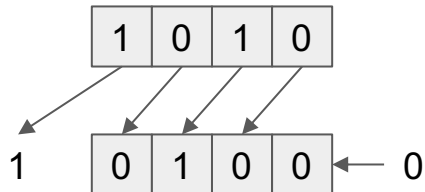
`s1 = t1 << 2;` `sll $s1, $t1, 2`

`s1 = t1 >>> 2;` `srl $s1, $t1, 2`

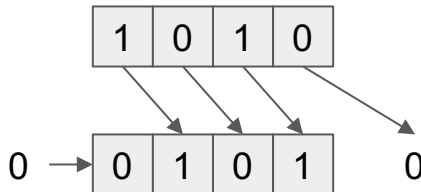
`s1 = t1 >> 2;` `sra $s1, $t1, 2`

- Let's Assume 4-bits and a shift of "1"

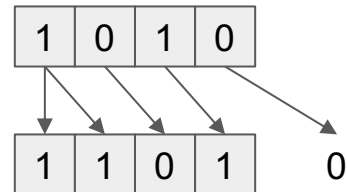
shift left logical



shift right logical



shift right arithmetic



Additional Shift-based Operations

- Rotates or Circular Shifts

- Rotate Left Logical
- Rotate Right Logical

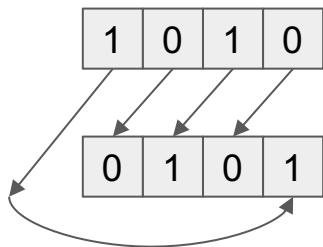
```
rol $s1, $t1, 2  
ror $s1, $t1, 2
```

```
sll $s1, $t1, 2  
srl $at, $t1, 30  
or $s1, $s1, $at
```

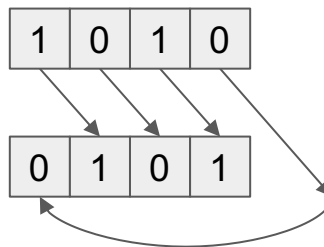
```
srl $s1, $t1, 2  
sll $at, $t1, 30  
or $s1, $s1, $at
```

- Typically, not supported in high-level languages
- Let's Assume 4-bits and a shift of "1"

rotate left



rotate right



Repositioning Fields within a Register

The blue cells represent a field within the register!

- Consider a register (16 bits) containing information
- Consider extracting a subrange of bits

