Bitwise Operations

- Bitwise operations in high-level languages are applied to integers
- Java has three primary sizes for integers
- Two types of Bitwise Operations
  - Boolean based operations
  - Shift-based operations

- *unsigned* short int (16 bit chunks)
- *unsigned* int (32 bit chunks)
- *unsigned* long int (64 bit chunks)

- 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`

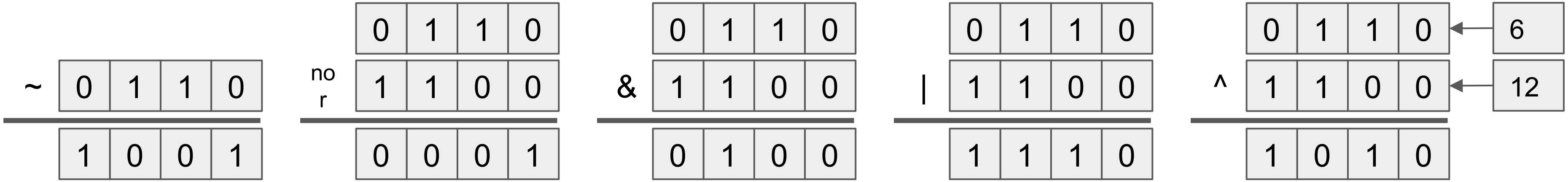
- Shift-based Operations:

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

Boolean-based Bitwise Operations

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

- Let's assume 4-bit chunks:
  - Complement: `s1 = ~ t1` `nor $s1, $t1, $zero`
  - And: `s1 = t1 & t2` `and $s1, $t1, $t2`
  - Or: `s1 = t1 | t2` `or  $s1, $t1, $t2`
  - Xor: `s1 = t1 ^ t2` `xor $s1, $t1, $t2`

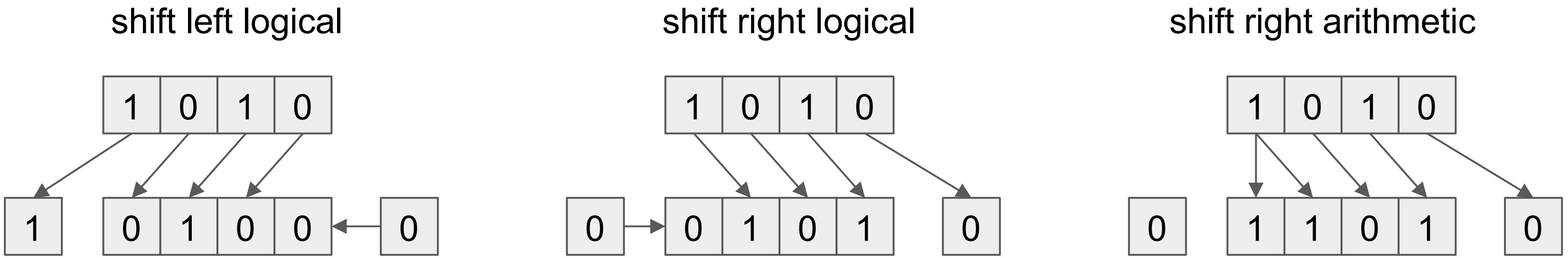


Shift-based Operations

- Java and MIPS supported:

Shift Left Logical	`s1 = t1 << 2`	`sll $s1, $t1, 2`
Shift Right Logical	`s1 = t1 >> 2`	`srl $s1, $t1, 2`
Shift Right Arithmetic	`s1 = t1 >>> 2`	`sra $s1, $t1, 2`
Shift Left Logical Variable	`s1 = t1 << t2`	`sllv $s1, $t1, $t2`
Shift Right Logical Variable	`s1 = t1 >> t2`	`srlv $s1, $t1, $t2`
Shift Right Arithmetic Variable	`s1 = t1 >>> t2`	`srav $s1, $t1, $t2`

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



shift left logical

shift right logical

shift right arithmetic

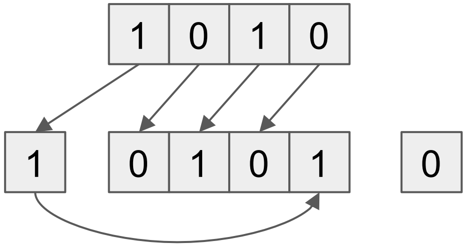
Additional Shift-based Operations

- Rotates or Circular Shifts
  - Rotate Left Logical `rol $s1, $t1, 2`
  - Rotate Right Logical `ror $s1, $t1, 2`
- Typically, not supported in high-level languages
- Let's Assume 4-bits and a shift of "1"

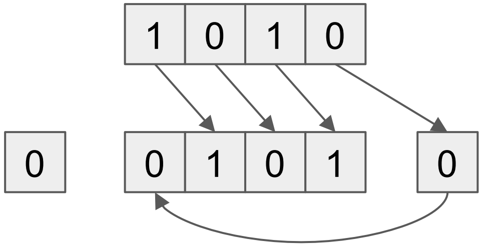
```
sll $s1, $t1, 2
srl $at, $t1, 30
or $s1, $s1, $at
```

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

rotate left

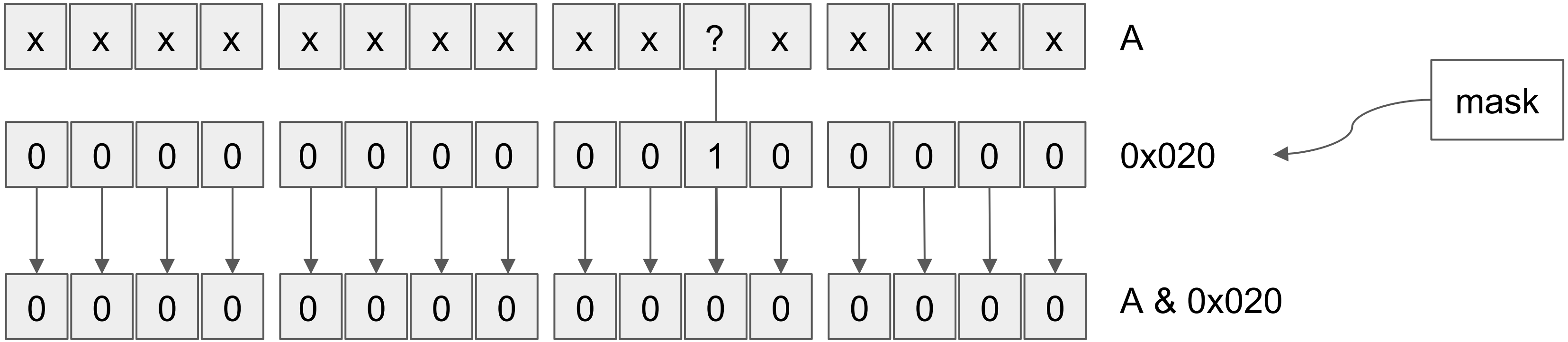


rotate right



Bit Manipulation: Testing the bit value

- Consider a register (16 bits) containing information
- Consider testing the value of a particular bit



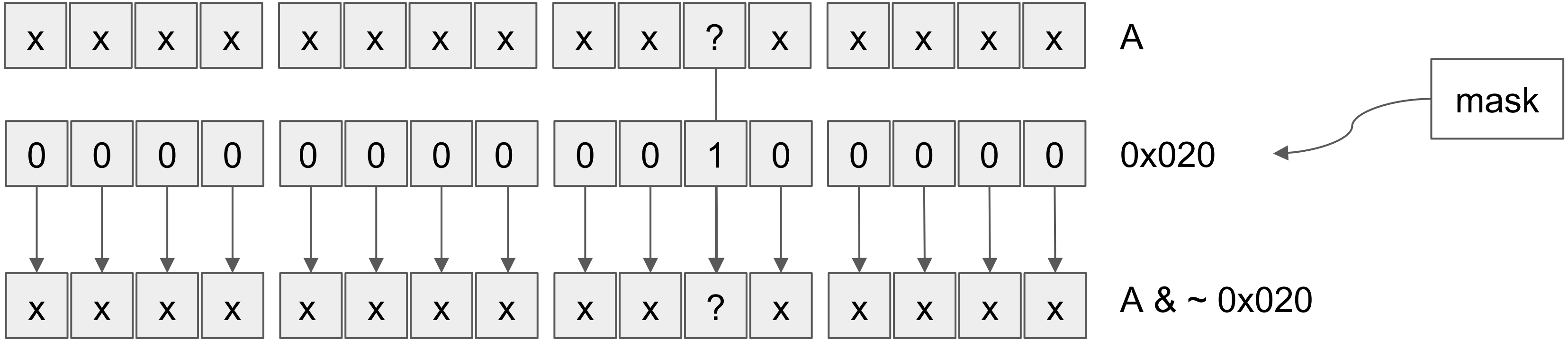
x	x	x	x	x	x	x	x	x	x	?	x	x	x	x	x	A
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0x020
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A & 0x020

- If the resulting value is equal to zero then

Z	S	O	C
?	0	0	0

Bit Manipulation: Clearing a bit

- Consider a register (16 bits) containing information
- Consider testing the value of a particular bit

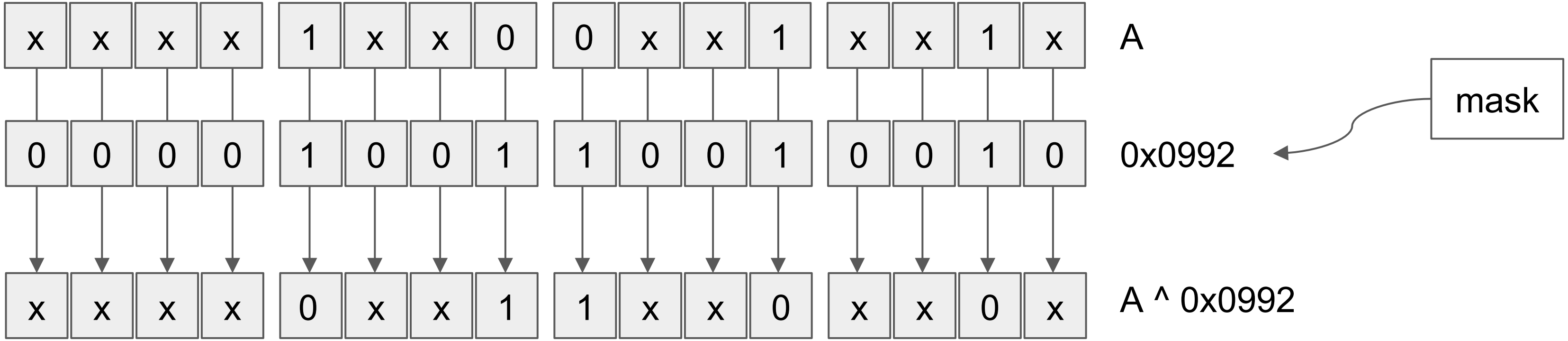


x	x	x	x	x	x	x	x	x	x	?	x	x	x	x	x	A
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0x020
x	x	x	x	x	x	x	x	x	x	?	x	x	x	x	x	A & ~ 0x020

- Native instruction on ARM: bic A, A #0x200

Bit Manipulation: Flipping the value of a set of bits

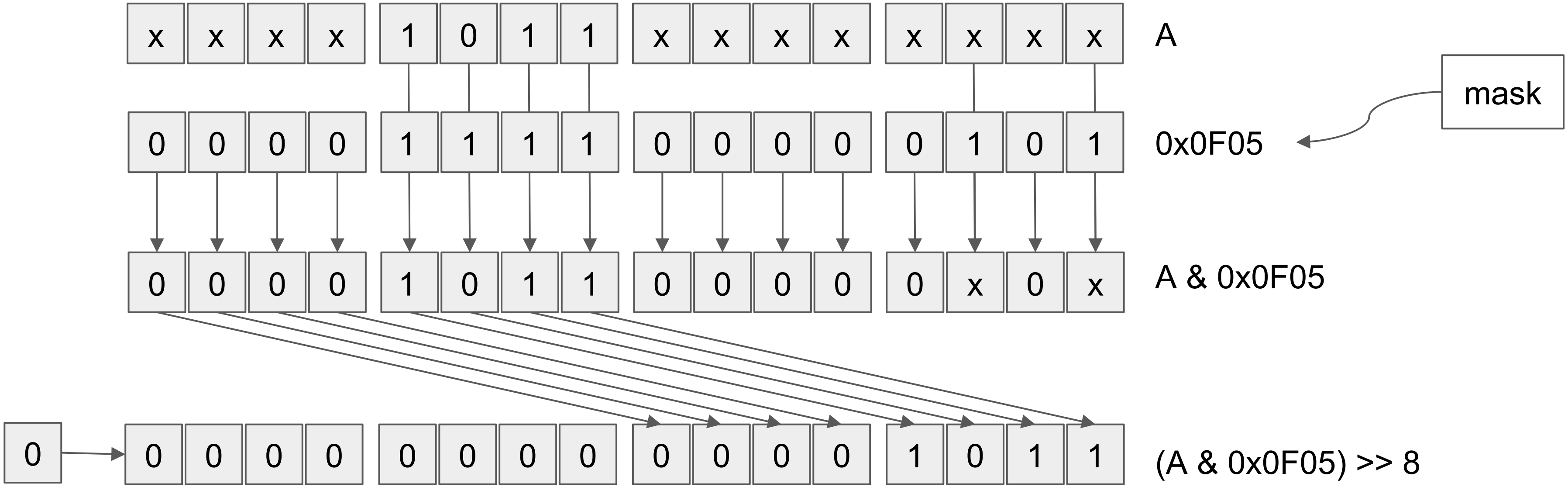
- Consider a register (16 bits) containing information
- Consider extracting a subrange of bits



x	x	x	x	1	x	x	0	0	x	x	1	x	x	1	x	A
0	0	0	0	1	0	0	1	1	0	0	1	0	0	1	0	0x0992
x	x	x	x	0	x	x	1	1	x	x	0	x	x	0	x	A ^ 0x0992

Bit Manipulation: Extracting a subrange of bits

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



	x	x	x	x	1	0	1	1	x	x	x	x	x	x	x	x	A
	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	0x0F05
	0	0	0	0	1	0	1	1	0	0	0	0	0	x	0	x	A & 0x0F05
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	(A & 0x0F05) >> 8