Bitwise logical operations

R-type

```
and $rd, $rs, $rt   # R[d] <- R[s] & R[t]        Bitwise AND
or  $rd, $rs, $rt   # R[d] <- R[s] | R[t]        Bitwise OR
nor $rd, $rs, $rt   # R[d] <- ~(R[s] | R[t])     Bitwise NOR
xor $rd, $rs, $rt   # R[d] <- R[s] ^ R[t]        Bitwise XOR
```

what about bitwise negation (complement)?

can use XOR with -1:

```
addi    $8,$0,0x42   # put 2's comp. hex 42 into register 8
```

what value is in register 0?

```
addi    $9,$0,-1     # put 2's comp. -1 into register 9
```

note sign bit extension for immediate value

```
xor     $10,$8,$9    # xor register 8 and 9, result in 10
```

$r8	0000	0000	0000	0000	0000	0000	0100	0010	42
$r9	1111	1111	1111	1111	1111	1111	1111	1111	-1
$r10	1111	1111	1111	1111	1111	1111	1011	1101	

example of fewer instruction types --> more instructions

also no NAND (negated AND)

Bitwise logical operations

I-type

```
andi $rt, $rs, immed  # R[t] <- R[s] & immed
ori  $rt, $rs, immed  # R[t] <- R[s] | immed
xori $rt, $rs, immed  # R[t] <- R[s] ^ immed
```

Bitwise AND immediate

Bitwise OR immediate

Bitwise XOR immediate

Semantics:

$R[t] \leftarrow R[s]\ op\ (0)^{16}::IR_{15\text{-}0}$

Note that immediate value is zero-extended

No NOR immediate

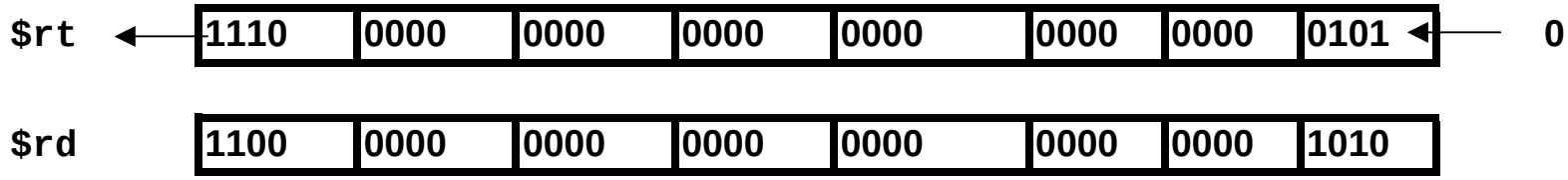
Bit shift operations: logical

R-type

b_{31-26}	b_{25-21}	b_{20-16}	b_{15-11}	b_{10-6}	b_{5-0}
opcode	$rs	$rt	$rd	shamt	function

logical shift: 0's shifted in

	# bits
left	
`sll $rd, $rt, shift_amt` `# R[d] <-- R[t] << shift_amt`	const
`sllv $rd, $rt, $rs` `# R[d] <-- R[t] << R[s]`	var



bits of $rt are shifted to the left by the number of bits given in shift_amt
or by value in register $rs
0's are shifted into least significant bit
if shift_amt, $rs is unused

What does this instruction do?

```
sll   $0, $0, 0
```

Bit shift operations: logical

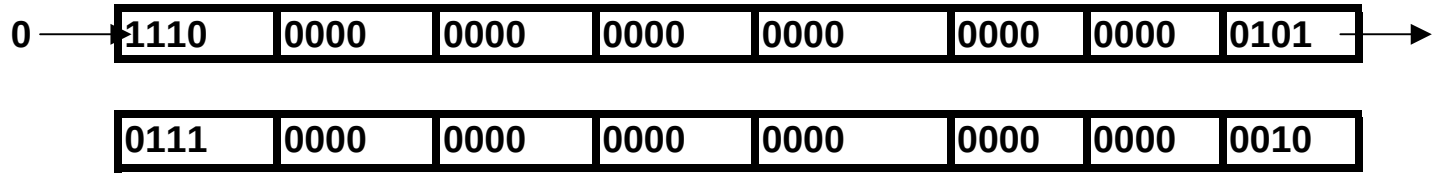
R-type

b_{31-26}	b_{25-21}	b_{20-16}	b_{15-11}	b_{10-6}	b_{5-0}
opcode	$rs	$rt	$rd	shamt	function

logical shift: 0's shifted in # bits

right

```
srl $rd, $rt, shift_amt     # R[d] <-- R[t] >> shift_amt      const
srlv $rd, $rt, $rs          # R[d] <-- R[t] >> R[s]           var
```



bits of $rt are shifted to the right by the number of bits given in shift_amt
or by value in register $rs
0's are shifted into most significant bit
if shift_amt, $rs is unused

What is the instruction above?

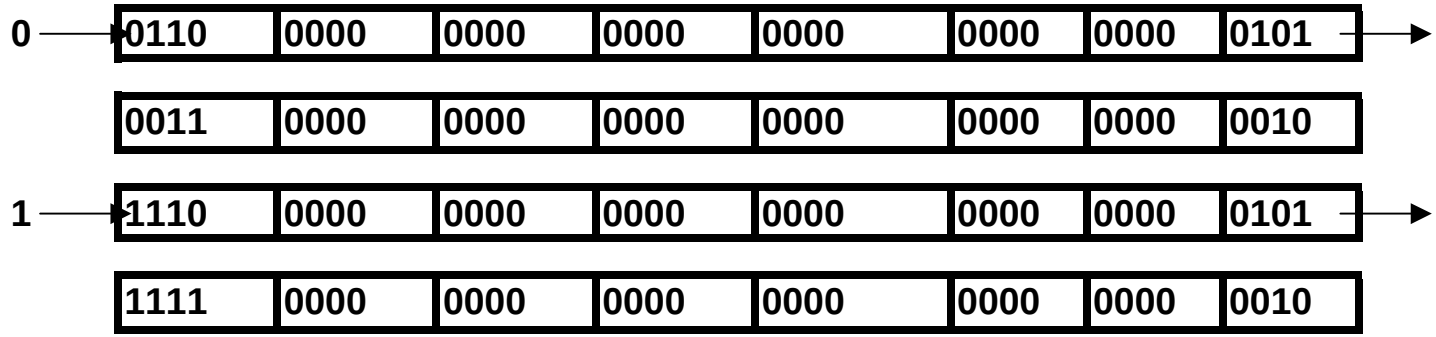
Bit shift operations: arithmetic

000000	00000	01001	01010	00011	000011
b_{31-26}	b_{25-21}	b_{20-16}	b_{15-11}	b_{10-6}	b_{5-0}
opcode	$rs	$rt	$rd	shamt	function

R-type

arithmetic shift: sign bit shifted in

		# bits
`sra $rd, $rt, shift_amt`	`# R[d] <-- R[t] >> shift_amt`	const
`srav $rd, $rt, $rs`	`# R[d] <-- R[t] >> R[s]`	var



bits of $rt are shifted to the right by the number of bits given in shift_amt
or by value in register $rs
sign bit (arithmetic) is shifted into most significant bit
if shift_amt, $rs is unused

why isn't there a sla or slav?

Bit shift operations: example

									hex	dec.
$r2										
$r3										
$r4										
$r5										
$r8	0000	0000	0000	0000	0000	0000	0100	0010	42	66
$r9										

```
example:
addi    $8,$0,0x42     # put two's comp. hex 42 into register 8
addi    $9,$0,5        # put two's comp. 5 into register 9
sll     $2,$8,3        # shift bits of reg 8 left by 3, result in reg 2
sllv    $3,$8,$9       # shift bits of reg 8 left by amount in reg 9,
                       #    result in reg 3
srl     $4,$8,3        # shift bits of reg 8 right by 3, result in reg 4
srlv    $5,$8,$9       # shift bits of reg 8 right by amount in reg 9,
                       #    result in reg 5
```

Bit shift operations: example

									hex	dec.
$r2										
$r3										
$r4										
$r5										
$r8	0000	0000	0000	0000	0000	0000	0100	0010	42	66
$r9	0000	0000	0000	0000	0000	0000	0000	0101	5	5

```
example:
addi    $8,$0,0x42     # put two's comp. hex 42 into register 8
addi    $9,$0,5        # put two's comp. 5 into register 9
sll     $2,$8,3        # shift bits of reg 8 left by 3, result in reg 2
sllv    $3,$8,$9       # shift bits of reg 8 left by amount in reg 9,
                         #    result in reg 3
srl     $4,$8,3        # shift bits of reg 8 right by 3, result in reg 4
srlv    $5,$8,$9       # shift bits of reg 8 right by amount in reg 9,
                         #    result in reg 5
```

Bit shift operations: example

								hex	dec.
$r2	0000	0000	0000	0000	0000	0010	0001	0000	210	528
$r3										
$r4										
$r5										
$r8	0000	0000	0000	0000	0000	0000	0100	0010	42	66
$r9	0000	0000	0000	0000	0000	0000	0000	0101	5	5

```
example:
addi    $8,$0,0x42     # put two's comp. hex 42 into register 8
addi    $9,$0,5        # put two's comp. 5 into register 9
sll     $2,$8,3        # shift bits of reg 8 left by 3, result in reg 2
sllv    $3,$8,$9       # shift bits of reg 8 left by amount in reg 9,
                       #    result in reg 3
srl     $4,$8,3        # shift bits of reg 8 right by 3, result in reg 4
srlv    $5,$8,$9       # shift bits of reg 8 right by amount in reg 9,
                       #    result in reg 5
```

Bit shift operations: example

									hex	dec.
$r2	0000	0000	0000	0000	0000	0010	0001	0000	210	528
$r3	0000	0000	0000	0000	0000	1000	0100	0000	840	2112
$r4										
$r5										
$r8	0000	0000	0000	0000	0000	0000	0100	0010	42	66
$r9	0000	0000	0000	0000	0000	0000	0000	0101	5	5

```
example:
addi    $8,$0,0x42     # put two's comp. hex 42 into register 8
addi    $9,$0,5        # put two's comp. 5 into register 9
sll     $2,$8,3        # shift bits of reg 8 left by 3, result in reg 2
sllv    $3,$8,$9       # shift bits of reg 8 left by amount in reg 9,
                       #    result in reg 3
srl     $4,$8,3        # shift bits of reg 8 right by 3, result in reg 4
srlv    $5,$8,$9       # shift bits of reg 8 right by amount in reg 9,
                       #    result in reg 5
```

Bit shift operations: example

									hex	dec.
$r2	0000	0000	0000	0000	0000	0010	0001	0000	210	528
$r3	0000	0000	0000	0000	0000	1000	0100	0000	840	2112
$r4	0000	0000	0000	0000	0000	0000	0000	1000	8	8
$r5										
$r8	0000	0000	0000	0000	0000	0000	0100	0010	42	66
$r9	0000	0000	0000	0000	0000	0000	0000	0101	5	5

```
example:
addi    $8,$0,0x42    # put two's comp. hex 42 into register 8
addi    $9,$0,5       # put two's comp. 5 into register 9
sll     $2,$8,3       # shift bits of reg 8 left by 3, result in reg 2
sllv    $3,$8,$9      # shift bits of reg 8 left by amount in reg 9,
                      #    result in reg 3
srl     $4,$8,3       # shift bits of reg 8 right by 3, result in reg 4
srlv    $5,$8,$9      # shift bits of reg 8 right by amount in reg 9,
                      #    result in reg 5
```

Bit shift operations: example

									hex	dec.
$r2	0000	0000	0000	0000	0000	0010	0001	0000	210	528
$r3	0000	0000	0000	0000	0000	1000	0100	0000	840	2112
$r4	0000	0000	0000	0000	0000	0000	0000	1000	8	8
$r5	0000	0000	0000	0000	0000	0000	0000	0010	2	2
$r8	0000	0000	0000	0000	0000	0000	0100	0010	42	66
$r9	0000	0000	0000	0000	0000	0000	0000	0101	5	5

```
example:
addi    $8,$0,0x42     # put two's comp. hex 42 into register 8
addi    $9,$0,5        # put two's comp. 5 into register 9
sll     $2,$8,3        # shift bits of reg 8 left by 3, result in reg 2
sllv    $3,$8,$9       # shift bits of reg 8 left by amount in reg 9,
                         #    result in reg 3
srl     $4,$8,3        # shift bits of reg 8 right by 3, result in reg 4
srlv    $5,$8,$9       # shift bits of reg 8 right by amount in reg 9,
                         #    result in reg 5
```

Arithmetic/logical: summary

		R-type	I-type
Arithmetic	Add unsigned	`addu`	`addiu`
	Add signed	`add`	`addi`
	Subtract unsigned	`subu`	
	Subtract signed	`sub`	
Boolean	AND	`and`	`andi`
	OR	`or`	`ori`
	Negated OR	`nor`	
	Exclusive OR	`xor`	`xori`
Shift (logical)	Left constant	`sll`	
	Right constant	`srl`	
	Left variable	`sllv`	
	Right variable	`srlv`	
Shift (arithmetic)	Right constant	`sra`	
	Right variable	`srav`	

Instruction Types

Arithmetic 

Logical 

Data Transfer

Compare/Branch

Jump