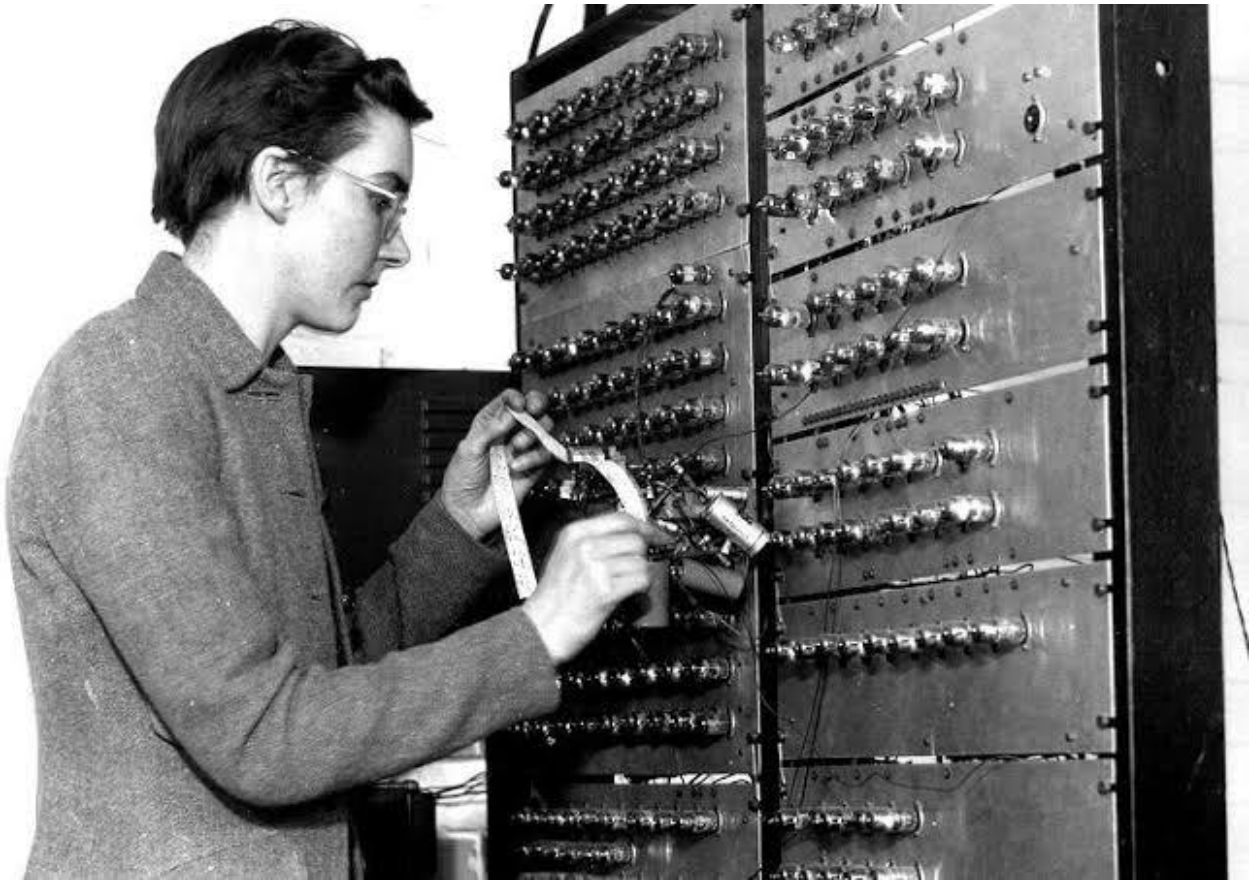
CSCI 210: Computer Architecture
Lecture 9: Logical Operations

Stephen Checkoway
Slides from Cynthia Taylor

Announcements

- Problem Set 2 due today
  - PS 3 available
- Lab 1 due Sunday
  - Lab 2 available later today

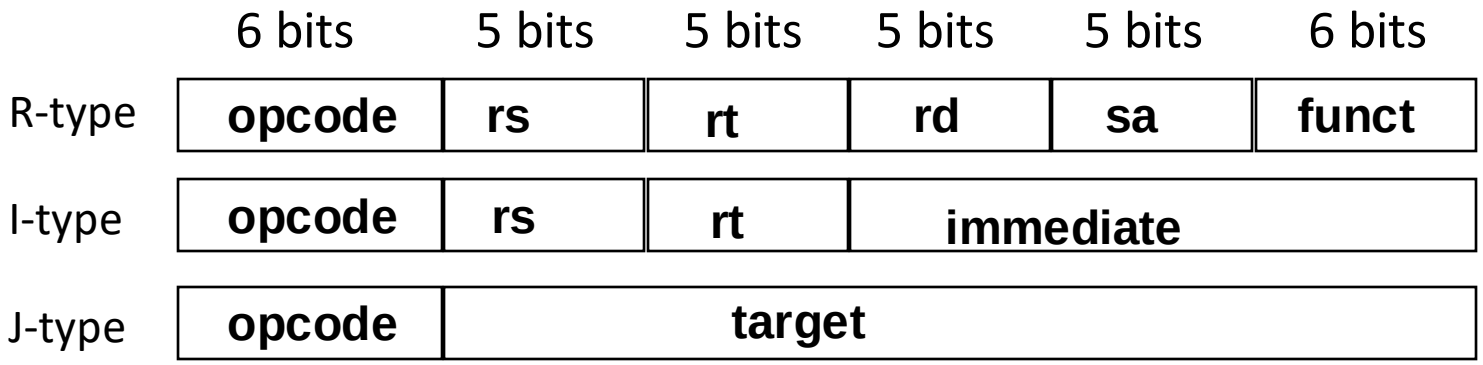
CS History: Kathleen Britton



- Applied mathematician and computer scientist
- Wrote the first assembly language and assembler in 1947
- Collaborated with Andrew Booth to develop three early computers: the ARC (Automatic Relay Calculator), SEC (Simple Electronic Computer), and APE(X)C
- Later worked with neural nets

Representing Instructions

- MIPS instructions
  - Encoded as 32-bit instruction words
  - Small number of formats encoding operation code (opcode), register numbers, …
  - Regularity!

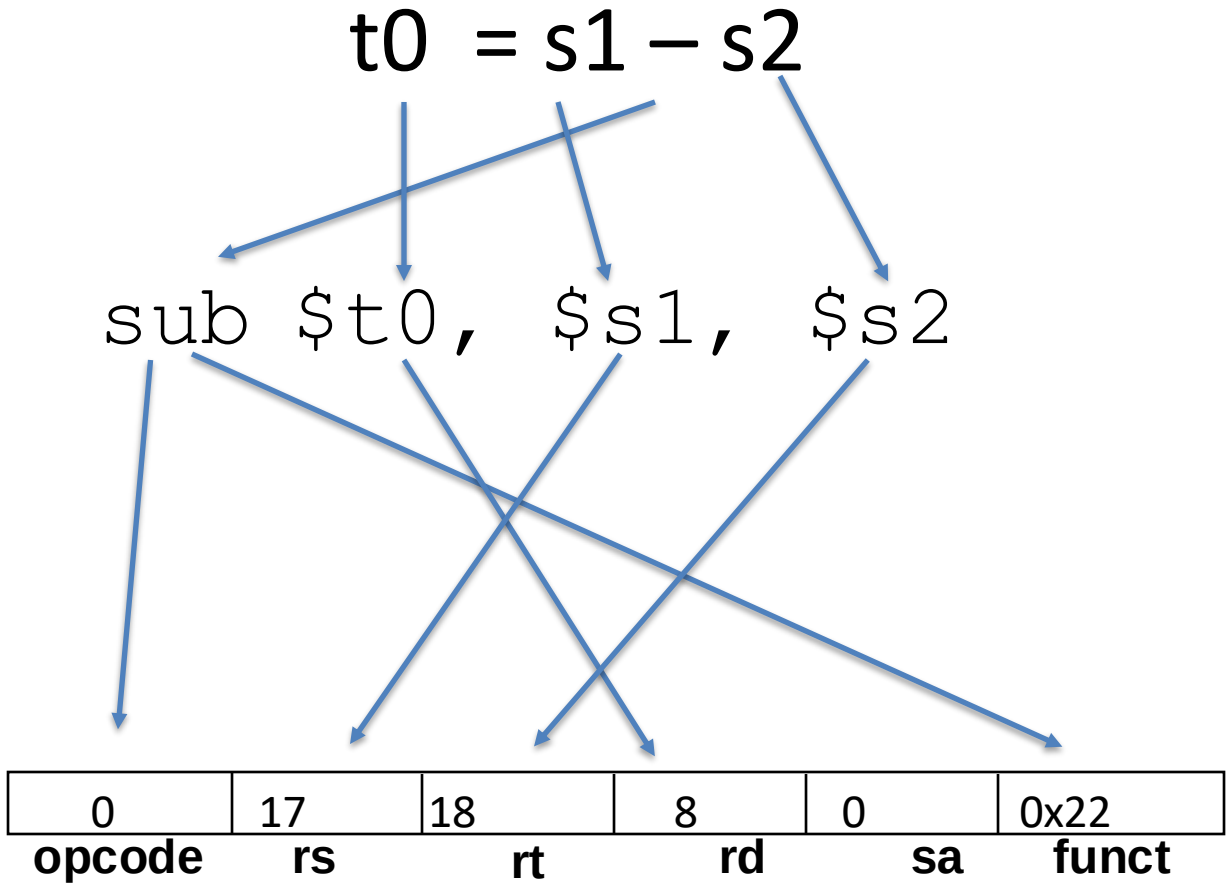


MIPS Instruction Fields

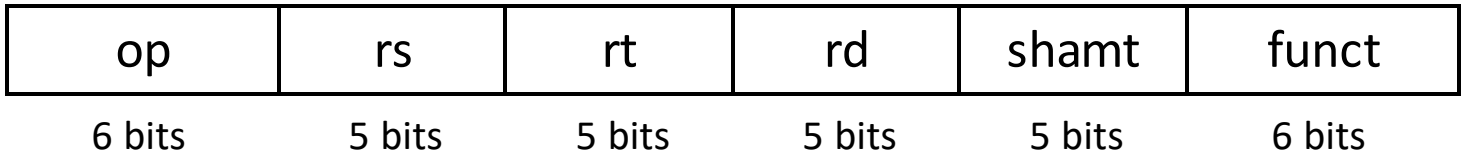
- MIPS fields are given names to make them easier to refer to

op	6-bits	opcode that specifies the operation
rs	5-bits	register file address of the first source operand
rt	5-bits	register file address of the second source operand
rd	5-bits	register file address of the result's destination
shamt	5-bits	shift amount (for shift instructions)
funct	6-bits	function code augmenting the opcode

MIPS Arithmetic Instructions Format



R-format Example




```
add $t0, $s1, $s2
```

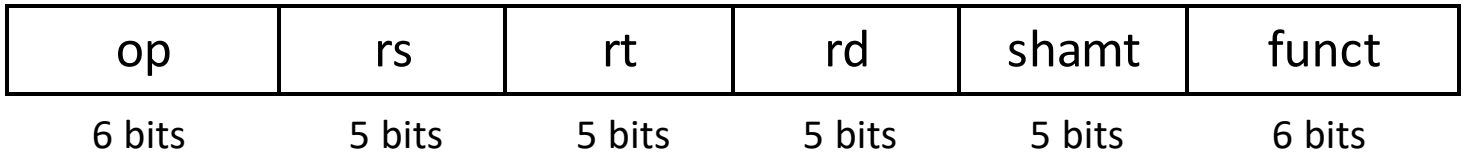
CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}

NAME	NUMBER	USE
$zero	0	The Constant Value 0
$at	1	Assembler Temporary
$v0-$v1	2-3	Values for Function Results and Expression Evaluation
$a0-$a3	4-7	Arguments
$t0-$t7	8-15	Temporaries
$s0-$s7	16-23	Saved Temporaries
$t8-$t9	24-25	Temporaries
$k0-$k1	26-27	Reserved for OS Kernel
$gp	28	Global Pointer
$sp	29	Stack Pointer
$fp	30	Frame Pointer
$ra	31	Return Address

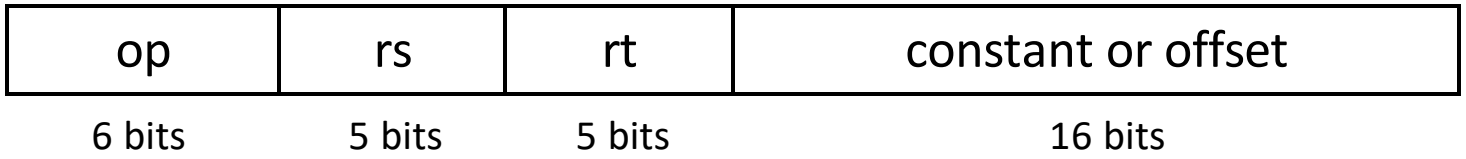
Convert this MIPS machine instruction to assembly:

000000 01110 10001 10010 00000 100010



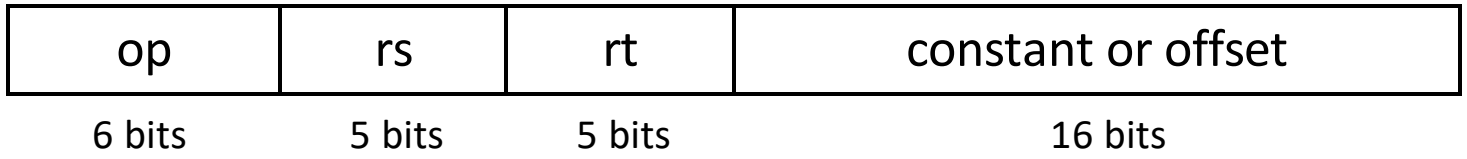
Selection	Instruction
A	add $s2, $t7, $s4
B	add $s1, $t6, $s3
C	sub $t6, $s1, $s2
D	sub $s2, $t6, $s1
E	None of the above

MIPS I-format Instructions



- Immediate arithmetic and load/store instructions
  - rt: destination or source register number
  - Constant: -2^{15} to $+2^{15} - 1$ (or 0 to $2^{16} - 1$ for some instructions)
  - offset: offset added to base address in rs

Machine Language – I Format



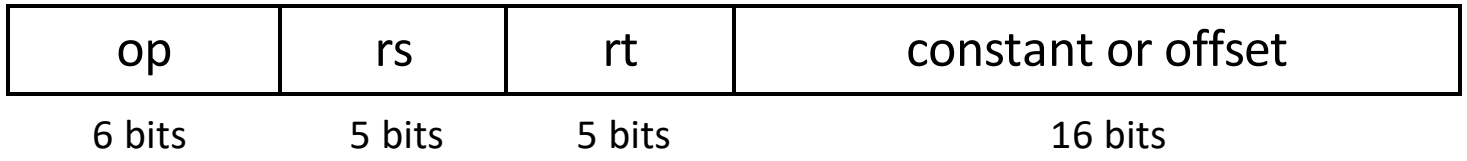
- Load/Store Instruction Format:

```
lw $t0, 24($s3)
```

Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		$0 / 27_{hex}$

NAME	NUMBER	USE
$zero	0	The Constant Value 0
$at	1	Assembler Temporary
$v0-$v1	2-3	Values for Function Results and Expression Evaluation
$a0-$a3	4-7	Arguments
$t0-$t7	8-15	Temporaries
$s0-$s7	16-23	Saved Temporaries
$t8-$t9	24-25	Temporaries
$k0-$k1	26-27	Reserved for OS Kernel
$gp	28	Global Pointer
$sp	29	Stack Pointer
$fp	30	Frame Pointer
$ra	31	Return Address

Machine Language – I Format



- Immediate Addition Instruction Format:

```
addi $t0, $s3, 26
```

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	`add`	R	R[rd] = R[rs] + R[rt]	(1)	$0 / 20_{hex}$
Add Immediate	`addi`	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	`addiu`	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	`addu`	R	R[rd] = R[rs] + R[rt]		$0 / 21_{hex}$

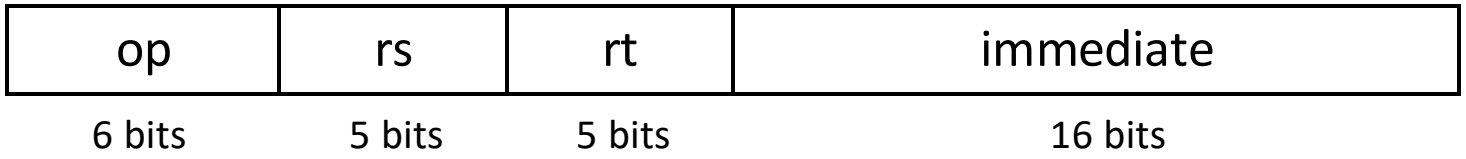
NAME	NUMBER	USE
$zero	0	The Constant Value 0
$at	1	Assembler Temporary
$v0-$v1	2-3	Values for Function Results and Expression Evaluation
$a0-$a3	4-7	Arguments
$t0-$t7	8-15	Temporaries
$s0-$s7	16-23	Saved Temporaries
$t8-$t9	24-25	Temporaries
$k0-$k1	26-27	Reserved for OS Kernel
$gp	28	Global Pointer
$sp	29	Stack Pointer
$fp	30	Frame Pointer
$ra	31	Return Address

Convert this MIPS assembly instruction to machine code

```
sw $t0, 32($s6)
```

Selection	Instruction
A	010101 11011 00100 0000 0000 0010 0000
B	101011 01000 10110 0000 0000 0010 0000
C	101011 10110 01000 0000 0000 0010 0000
D	000000 00010 00000 1010 1110 1100 1000
E	None of the above

Sign-extend vs. zero-extend



- The immediate field of an I-format instruction is either sign-extended or zero-extended
  - sign extension: the sign bit (bit 15) is copied into bits 31–16
  - zero extension: 0 is placed into bits 31–16
- Opcode determines which occurs

Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		$0 / 21_{hex}$
And	and	R	R[rd] = R[rs] & R[rt]		$0 / 24_{hex}$
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}

Questions about Machine Instructions?

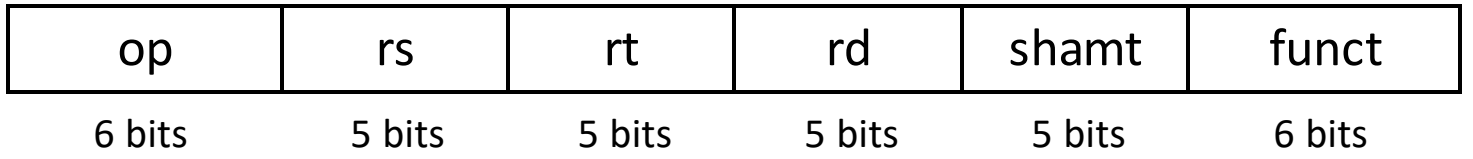
Logical Operations

- Instructions for bitwise manipulation

Operation	C	Java	MIPS
Shift left	<<	<<	`sll`
Shift right	>>	>>>	`srl`
Bitwise AND	&	&	`and, andi`
Bitwise OR	|	|	`or, ori`
Bitwise NOT	~	~	`nor`

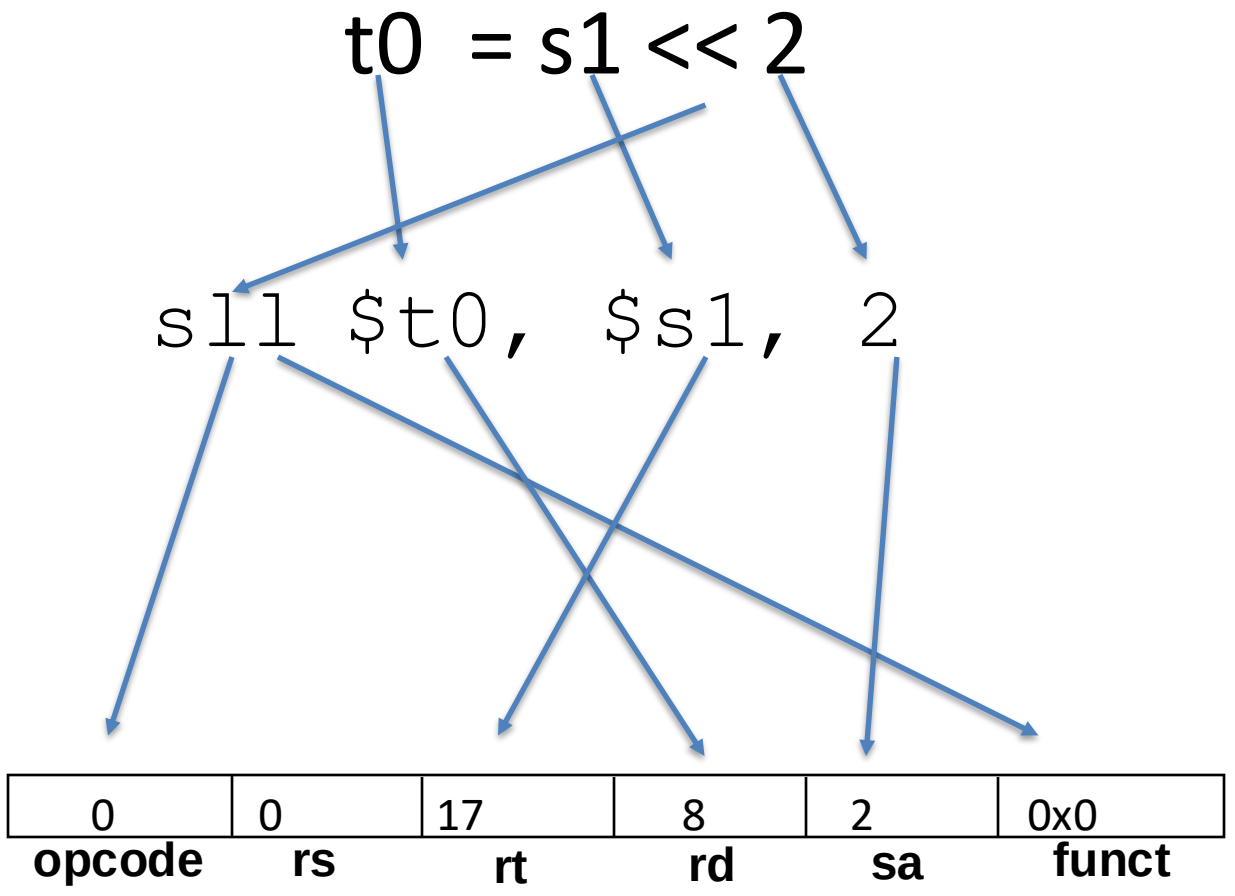
- Useful for extracting and inserting groups of bits in a word

Shift Operations



- shamt: how many positions to shift
- Shift left logical
  - Shift left and fill with 0 bits
  - sll by n bits multiplies by 2^n
- Shift right logical
  - Shift right and fill with 0 bits
  - srl by n bits divides by 2^n (unsigned only)

MIPS shift instructions



Shift left logical

- 0110 1001 << 2 in 8 bits
  - Most significant 2 bits are dropped
  - 2 0s are added to become the least significant bits
  - Result: ~~01~~ 1010 0100 => 1010 0100

Shift right logical

- 1010 1001 >>> 3 in 8 bits
  - Least significant 3 bits are dropped
  - 3 0s are added to become the most significant bits
  - Result: 0001 0101 ~~001~~ => 0001 0101

Shift right arithmetic

- sra rd, rt, shamt
  - Shift right and copy the sign bit
- 1010 1001 >> 3 in 8 bits
  - Least significant 3 bits are dropped
  - 3 1s are added because the MSB is 1 to become the most significant bits
  - Result: 1111 0101 ~~001~~ => 1111 0101

A new op HEXSHIFTRIGHT shifts hex numbers right by a digit. HEXSHIFTRIGHT i times is equivalent to

A. Dividing by i

B. Dividing by 2^i

C. Dividing by 16^i

D. Multiplying by 16^i

Remember Boolean Operations?

- and, or, not . . .
- Now we'll apply them to bits!
- Just think of 1 as True, and 0 as False

And Truth Table

	0	***1***
0	0	0
1	0	1

AND Operations

- Useful to mask bits in a word
  - Select some bits, clear others to 0

```
and $t0, $t1, $t2
```

$t2	0000 0000 0000 0000 0000 1101 1100 0000
$t1	0000 0000 0000 0000 0011 1100 0000 0000
$t0	0000 0000 0000 0000 0000 1100 0000 0000

AND identities (for a single bit)

- x & 0 =
- x & 1 =

01101001 & 11000111

A. 00010000

B. 01000001

C. 10101110

D. 11101111

If we want to zero out bits* 3 – 0 in a byte we should AND with

A. 00000000

B. 00001111

C. 11110000

D. 11111111

*MSB is on the left, rightmost bit is 0

Or Truth Table

	0	***1***
0	0	1
1	1	1

OR Operations

- Useful to set bits in a word
  - Set some bits to 1, leave others unchanged

```
or $t0, $t1, $t2
```

$t2	0000 0000 0000 0000 0000 1101 1100 0000
$t1	0000 0000 0000 0000 0011 1100 0000 0000
$t0	0000 0000 0000 0000 0011 1101 1100 0000

OR Identities (for a single bit)

- x | 0 =
- x | 1 =

01101001 | 11000111

A. 00010000

B. 01000001

C. 10101110

D. 11101111

Reading

- Next lecture: Branching instructions
  - Read Section 2.8

- Problem Set 2 due today

- Lab 1 due on Monday