

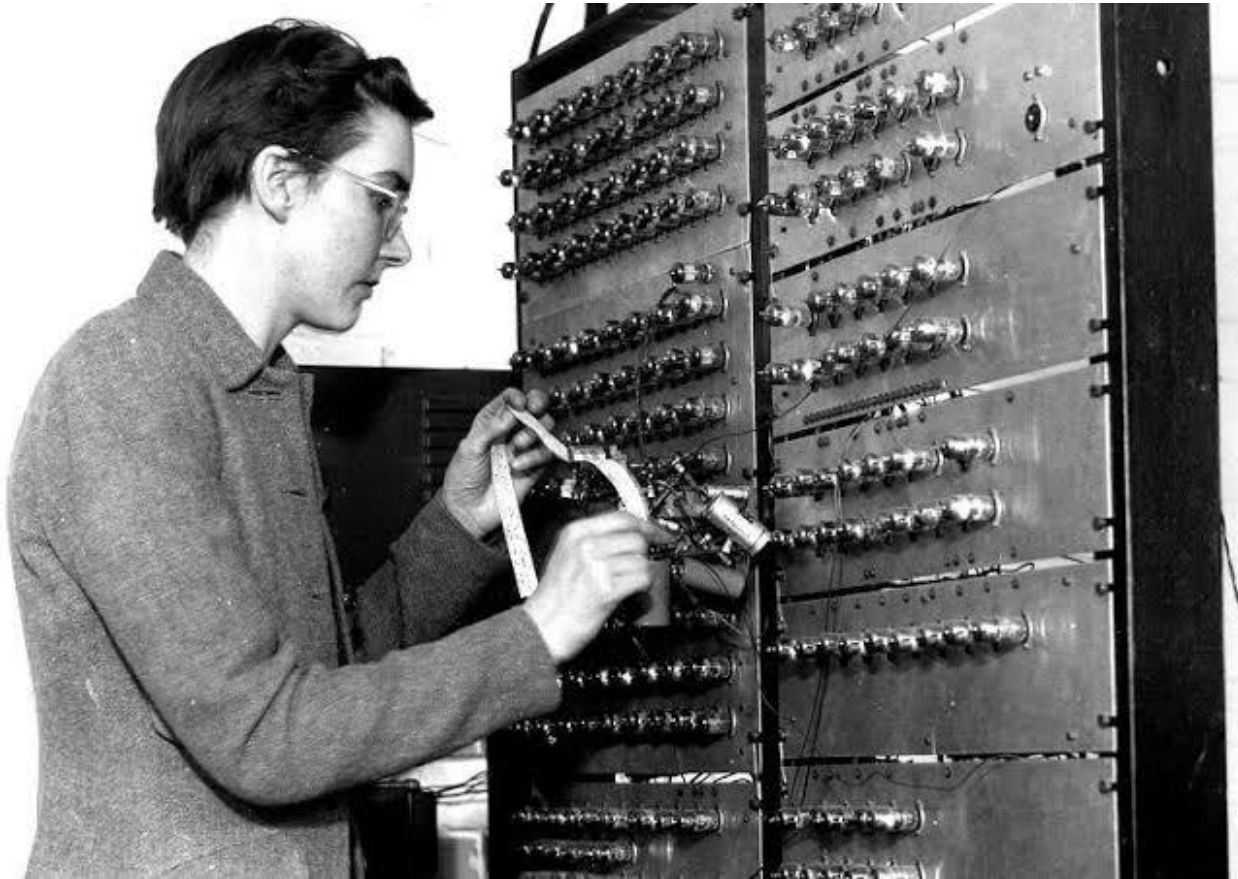
# CSCI 210: Computer Architecture

## Lecture 9: Logical Operations

Stephen Checkoway

Slides from Cynthia Taylor

# 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

# 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

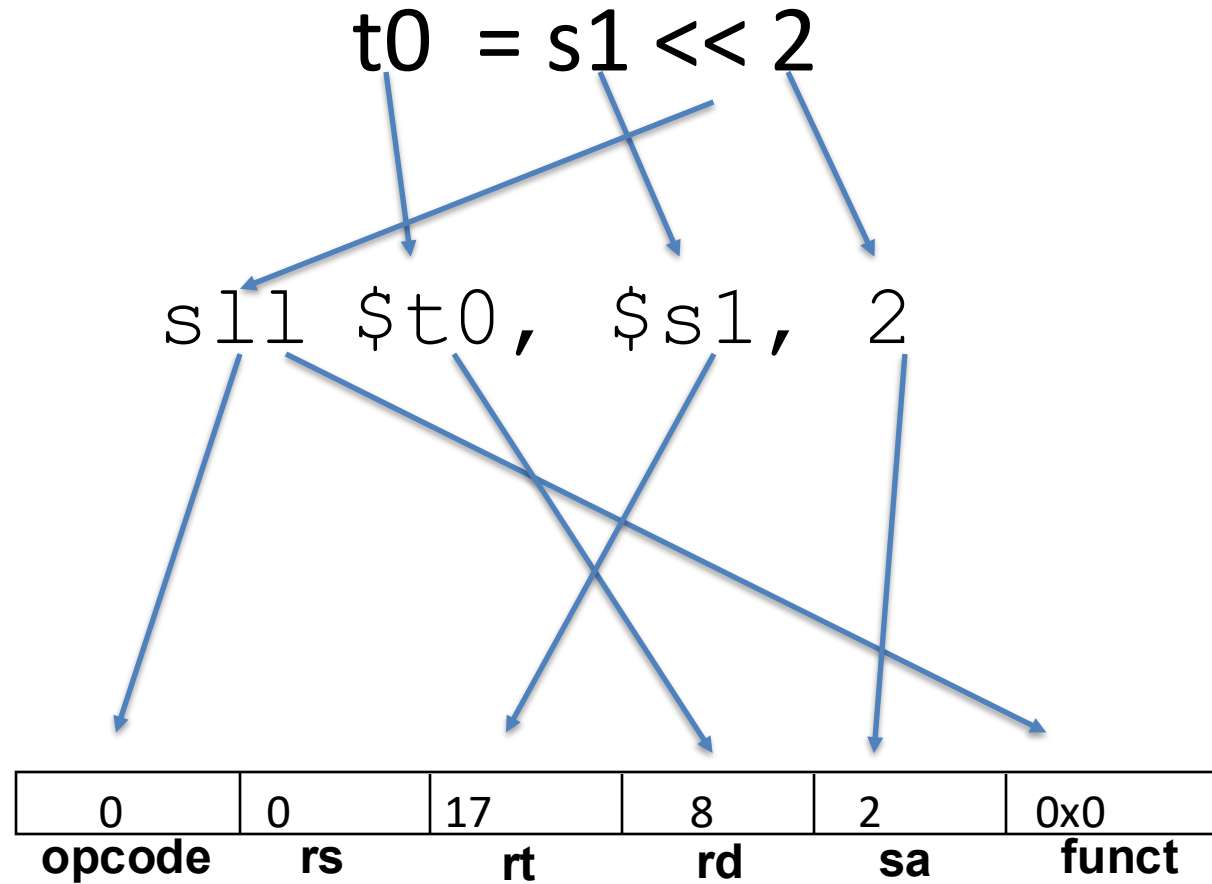
- Useful for extracting and inserting groups of bits in a word

# 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 01**00** => 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: **000**1 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

	<i><b>0</b></i>	<i><b>1</b></i>
<i><b>0</b></i>	0	0
<i><b>1</b></i>	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)

- $x \& 0 =$
- $x \& 1 =$

$$\begin{array}{r} 01101001 \\ \& 11000111 \end{array}$$

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 (bit 7) is on the left,  
LSB (bit 0) is on the right

One way to represent colors is to specify the amount of **red**, **green**, and **blue** (RGB) that makes up the color. The three color “channels” are often packed into a 32-bit integer as follows:

00000000 **rrrrrrrr** **gggggggg** **bbbbbbbb**

where each color is represented in 8 bits. If `color` is a 32-bit RGB color, which high-level expression extracts the green channel as an 8-bit value in the range 0–255?

A. `green = (color >> 8) & 0xFF`

B. `green = (color >> 16) & 0xFF`

C. `green = (color & 0xFF) >> 8`

D. `green = (color & 0xFF00) >> 16`

E. More than one of the above (which?)



Assume `color` is stored in `$t0` and `green` should be extracted to `$t1`. Which sequence of MIPS instructions corresponds to  $\text{green} = (\text{color} \gg 8) \ \& \ 0\text{xFF}$

A. `sll $t1, $t0, 8`  
    `andi $t1, $t1, 0xFF`

B. `srl $t1, $t0, 8`  
    `andi $t1, $t1, 0xFF`

C. `andi $t1, $t0, 0xFF`  
    `sll $t1, $t1, 8`

D. `andi $t1, $t0, 0xFF`  
    `srl $t1, $t1, 8`

# Or Truth Table

	<i><b>0</b></i>	<i><b>1</b></i>
<i><b>0</b></i>	0	1
<i><b>1</b></i>	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)

- $x \mid 0 =$

- $x \mid 1 =$

01101001  
| 11000111

A. 00010000

B. 01000001

C. 10101110

D. 11101111

Recall RGB:

00000000 rrrrrrrr gggggggg bbbbbbbb

If  $r$ ,  $g$ , and  $b$  are values in the range 0–255, how can we construct the RGB value,  $c$ , whose channels are  $r$ ,  $g$ , and  $b$ ?

A.  $c = (r \ll 16) \mid (g \ll 8) \mid b$

B.  $c = (r \ll 24) \mid (g \ll 16) \mid (b \ll 8)$

C.  $c = (r \gg 8) \mid (g \gg 16) \mid (b \gg 24)$

D.  $c = (r \gg 16) \mid (g \gg 24) \mid (b \gg 32)$

E. More than one of the above (which?)

# Nor Truth Table

	<i><b>0</b></i>	<i><b>1</b></i>
<i><b>0</b></i>	1	0
<i><b>1</b></i>	0	0

# NOR Operations

- MIPS has NOR 3-operand instruction
  - $a \text{ NOR } b = \text{NOT } (a \text{ OR } b)$

```
nor $t0, $t1, $t2
```

\$t2    0000 0000 0000 0000 0000 1101 1100 0000

\$t1    0000 0000 0000 0000 0011 1100 0000 0000

\$t0    1111 1111 1111 1111 1100 0010 0011 1111



01101001  
NOR 11000111

A. 00010000

B. 01000001

C. 10101110

D. 11101111

# NOT operations

- Inverts all the bits in a word
  - Change 0 to 1, and 1 to 0

MIPs does not need a NOT operation because we can use \_\_\_\_\_ for NOT \$t1, \$t2

A. `nor $t1, $t2, $zero`

B. `nor $t1, $t2, $t3`, where all bits in \$t3 are set to 1

C. `nori $t1, $t2, 0b11111111_11111111`, where nori is Nor Immediate

D. It does require a NOT operation

E. None of the above are correct

# XOR Truth Table

	<i><b>0</b></i>	<i><b>1</b></i>
<i><b>0</b></i>	0	1
<i><b>1</b></i>	1	0

# XOR Operations

- Exclusive OR (written  $x \oplus y$  or  $x \wedge y$ )
  - Set bits to one only if they are not the same

`xor $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 0001 1100 0000

01101001 XOR 11000111

A. 00010000

B. 01000001

C. 10101110

D. 11101111

# XOR Identities (for a single bit)

- $x \text{ XOR } 0 =$
- $x \text{ XOR } 1 =$

10 & 7

- A. 0
- B. 2
- C. 7
- D. 10
- E. None of the above



Set bit 4 in byte x to 1, leaving the rest of the bits unchanged

A.  $x = x \text{ AND } 00010000$

B.  $x = x \text{ AND } 11101111$

C.  $x = x \text{ OR } 00010000$

D.  $x = x \text{ NOR } 11101111$

# Invert bits 2–0 of x

A.  $x = x \text{ AND } 00000111$

B.  $x = x \text{ OR } 00000111$

C.  $x = x \text{ NOR } 00000111$

D.  $x = x \text{ XOR } 00000111$

Find the ones' complement of  $x$  (in 8 bits)

A.  $x \text{ XOR } 00000000$

B.  $x \text{ XOR } 11111111$

C.  $x \text{ XOR } 11111110$

D.  $x \text{ OR } 11111111$

# Reading

- Next lecture: Branching instructions