Bitwise Logic Operations

2-21-2003

Opening Discussion

- What did we talk about last class?
- What different logic operators do you know about? What do they operate on? What is implied by saying bitwise logic operators?
- Too many of you seem lost. How many of you have honestly read 100 pages of the book? You need to communicate confusion, but you also need to read enough and listen enough to be able to.

Logic Operations

- In your C++ programming you have used logic operators quite a bit. However, you have mainly been using the short circuit Boolean operators && and ||. (You might think about what that means in assembly language.)
- We want to look at these and other logic operators as more mathematical operations on numbers now. In this mode they act on them bit by bit.

Bit Shifts

- When working with binary numbers, one of the things that you might want to do is to shift them.
- A left shift (<< in C, sll in MIPS assembly) moves all the bits to the left the specified number of positions. This basically multiplies by a power of two.
- A right shift (>> in C, srl in MIPS assembly) moves all the bits to the right the specified number of positions. This basically divides by a power of two.

The Shift Field

- Remember the structure of the R-type instructions? There was a field that we didn't use when we first came across them, it was the shamt field. Those 5 bits are used for these shift instructions. That is enough because shifting more than 32 bits doesn't make sense.
- Because they only take two registers, the rs field goes unused in these instructions.

AND

- A bitwise AND does what you are used to && doing, but it does it on each and every bit in the 32 bit number (it is & in C/C++). The result is a 32 bit number that only has bits on where the bits were on in both operands.
- This instruction has both a register and an immediate form. Obviously, the immediate form can only alter the lowest 16 bits of any number.

OR

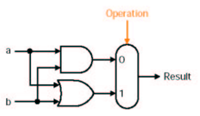
- A bitwise OR does what you are used to || doing, but it does it on each and every bit in the 32 bit number (it is | in C/C++). The result is a 32 bit number that has bits on where bits were on in either of the operands.
- This instruction has both a register and an immediate form. Obviously, the immediate form can only alter the lowest 16 bits of any number.

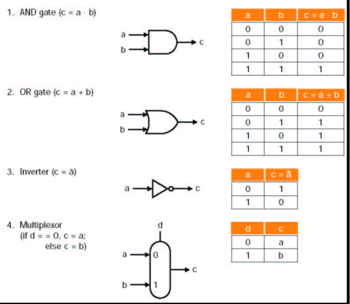
XOR and NOR

- MIPS assembly also provides XOR and NOR operations. XOR (exclusive or, ^ in C/C++) turns a bit on if one of the operand bits is on, but not both. NOR is short for NOT OR and is on only if both operand bits are off.
- XOR has both register and immediate forms. NOR only has a register form.

A 1-bit ALU

- We build all of the operations we want from basic one bit logic operations.





a	b	c = a · b
0	0	0
0	1	0
1	0	0
1	1	1

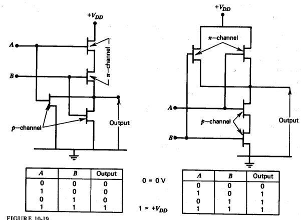
a	b	c = a + b
0	0	0
0	1	1
1	0	1
1	1	1

a	c = ā
0	1
1	0

d	c
0	a
1	b

How do we build them?

- This figure we saw previously shows how they could build a single bit AND or OR into a chip with just transistors.



A	*B*	Output
0	0	0
1	0	0
0	1	0
1	1	1

A	*B*	Output
0	0	0
1	0	1
0	1	1
1	1	1

FIGURE 10-19
MOSFET logic circuits, showing (*a*) an *A and B* circuit, and (*b*) an *A or B* circuit.

Minute Essay

- Convert to binary, then do the operation for 145 xor 78.
- Remember to turn in your assignment #3 before you leave.
- I won't be in town Monday and Tuesday so there will be no class on Monday. Hopefully that will lead to you getting some graded assignments back on Wednesday.