Bitwise Operators

Instructor: Dr. Vinicius Prado da Fonseca (vpradodafons@online.mun.ca)

Bitwise operations

- Bitwise or bit-level operations
- Section 4.2.1 of the textbook
- Some topics from now on will be found in chapter 4
- Mentioned on D2L notes

Bitwise operations

- Bitwise operators act on single-bit signals or on multi-bit busses.
- Can be affected by the endianness
- Little-endian order (used during this course)
    - the least significant bit has the smallest index number.
        - A = 01001
        - A[0] => 0100[1]
- Big-endian order
    - the most significant bit has the smallest index number
        - A= 01001
        - A[0] => [0]1001
- Endianness is arbitrary, each system uses one but we will use little-endian for now on, unless stated the opposite.

Single Bit Boolean

- NOT: a = 0; ~a=1
- AND: 0 & 0 = 0; 1 & 1 = 1; ...
- OR: 0 | 0 = 0; 0 | 1 = 1; ...
- XOR: 0 ^ 0 = 0; 0 ^ 1 = 1; ...

Bitwise operations - Examples

- Inverter (~)
  - a = 0; ~a=1
  - a[3:0] -> ~a[3:0]

    a 0010

    ~a
- AND (&)
  - 0 & 0 = 1; 1 & 1 = 1; ...
  - a = 01101; b = 11001
  - a & b = ?

    a 01101

    b 11011

    a & b
- OR (|)
  - 0 | 0 = 0; 0 | 1 = 1; ...
  - a = 01101; b = 11001
  - a | b = ?

    a 01101

    b 11011

    a | b
- XOR (^)
  - 0 ^ 0 = 0; 0 ^ 1 = 1; ...
  - a = 01101; b = 11001
  - a ^ b = ?

    a 01101

    b 11011

    a ^ b

Bitwise operations - Examples

- Inverter (~)
  - a = 0; ~a=1
  - a[3:0] -> ~a[3:0]

a	0010
~a	1101

- AND (&)
  - 0 & 0 = 1; 1 & 1 = 1; ...
  - a = 01101; b = 11001
  - a & b = ?

a	01101
b	11011
a & b	01001

- OR (|)
  - 0 | 0 = 0; 0 | 1 = 1; ...
  - a = 01101; b = 11001
  - a | b = ?

a	01101
b	11011
a | b	11111

- XOR (^)
  - 0 ^ 0 = 0; 0 ^ 1 = 1; ...
  - a = 01101; b = 11001
  - a ^ b = ?

a	01101
b	11011
a ^ b	10110

Boolean operators not defined for floating

- In general, the boolean operators can not be used with floating point numbers
- Python:

```
Traceback (most recent call last):
  File "floating.py", line 1, in <module>
    print( 3.14 & 0xff )
TypeError: unsupported operand type(s) for &: 'float' and 'int'
```

Shifting

- Most computers and programming languages support shifting the bits in the integer variables.
- The bits can be shifted left or right.
- Right shifting usually has three types:
  - Logic right shift:
    - Zeros are shifted in.
  - Rotational right shift:
    - LSB is shifted in.
  - Arithmetic right shift:
    - The sign bit is retained, and its value is copied when shifted.
- The shifting operators are:
  - a << b // shift the bits in **a** to the left by **b** bits
  - a >> b // shift the bits in **a** to the right by **b** bits

Shifting - Examples

- Left-shift
  - a = 0000 0101
  - a<<1

a	0000 0101	(5_{10})
a<<1		

  - a<<2

a	0000 0101	(5_{10})
a<<2		

  - a<<3

a	0000 0101	(5_{10})
a<<3		

- Right-shift (logic)
  - a = 0000 1010
  - a>>1

a	0000 1010	(10_{10})
a>>1		

  - a = 0000 0101
  - a>>2

a	0000 0101	(5_{10})
a>>2		

  - a = 0001 0000
  - a>>3

a	0001 0000	(16_{10})
a>>3		

Shifting - Examples

- Left-shift
  - a = 0000 0101
  - a<<1

a	0000 0101	(5_{10})
a<<1	0000 1010	(10_{10}) x2

  - a<<2

a	0000 0101	(5_{10})
a<<2	0001 0100	(20_{10}) x4

  - a<<3

a	0000 0101	(5_{10})
a<<3	0010 1000	(40_{10}) x8

- Right-shift (logic)
  - a = 0000 1010
  - a>>1

a	0000 1010	(10_{10})	
a>>1	0000 0101	(5_{10})	/2

  - a = 0000 0101
  - a>>2

a	0000 0101	(5_{10})	
a>>2	0000 0001	(1_{10})	/4

  - a = 0001 0000
  - a>>3

a	0001 0000	(16_{10})	
a>>3	0000 0010	(2_{10})	/8

Shifting - Examples

- Left-shift (multiplication by 3)
  - a = 0000 0101
  - r = a * 3

    ?

MEMORIAL UNIVERSITY
www.mun.ca

Shifting - Examples

- Left-shift (multiplication by 3)
  - a = 0000 0101
  - r = a * 3
  - r = a<<1

a	0000 0101	(5_{10})
r = a<<1	0000 1010	(10_{10}) x2

  - r = r+a

r	0000 1010	(10_{10})
a	0000 0101	(5_{10})
r = r+a	0000 1111	(15_{10}) x3

Shifting - Examples

- Right-shift (sequential subtractions)
  - a = 0000 1111
  - a/3
  - r=?

Shifting - Examples

- Right-shift (counting sequential subtractions)
  - a = 0000 1111
  - a/3
  - r=0

```
r=r+1=1     0000 1111  (15₁₀)
            1111 1101  (-3₁₀)
r=r+1=2     0000 1100
            1111 1101  (-3₁₀)
r=r+1=3     0000 1001
            1111 1101  (-3₁₀)
r=r+1=4     0000 0110
            1111 1101  (-3₁₀)
r=r+1=5     0000 0011
            1111 1101  (-3₁₀)
r = 5       0000 0000
```

Shifting - Examples

- Arithmetic Right-shift
  - a = 1111 1010 (-6_{10})   0000 0110 -> 1111 1010
  - a>>1

a	1111 1010	(-6_{10})
a>>1	1111 1101	(-3_{10}) /2

- Rotational right-shift
  - a = 1011 0101
  - a >>1

a	1011 0101
a>>1	?
a>>1	?

Shifting - Examples

- Arithmetic Right-shift
  - a = 1111 1010 (-6_{10})   0000 0110 -> 1111 1010
  - a>>1

a	1111 1010	(-6_{10})
a>>1	1111 1101	(-3_{10}) /2

- Rotational right-shift
  - a = 1011 0101
  - a >>1

a	1011 0101
a>>1	1101 1010
a>>1	0110 1101

MEMORIAL UNIVERSITY
www.mun.ca

Extracting Bits

A combination of boolean operators and shifting can be used to extract bits from an integer: “a” is the number, “t” is a temporary variable, “r” is the result.

```
a    0110 1101 (extract the 4 LSB)
```

Extracting Bits

A combination of boolean operators and shifting can be used to extract bits from an integer:

```
a     0110 1101
t     0000 1111
a&t   0000 1101
```

To extract the rest

```
a>>4 0000 0110
 a&t 0000 0110
```

Replacing Bits

A combination of boolean operators and shifting can also be used to replace bits in an integer.

Replace the three lower bits from “a” to another variable “r”

```
a     0110 1001
```


MEMORIAL UNIVERSITY
www.mun.ca

Replacing Bits

A combination of boolean operators and shifting can also be used to replace bits in an integer.

Replace the three lower bits from a to another variable r

```
a     0110  1001
t     0000  0111
~t    1111  1000
v     0000  0110

r = a&~t  0110 1000
r = r|v   0110 1110
```

Counting Bits

A combination of boolean operators and shifting can be used to extract bits from an integer:

a 0000 1101

MEMORIAL UNIVERSITY
www.mun.ca

Counting Bits

A combination of boolean operators and shifting can be used to extract bits from an integer:

a	0000 1101	
t	0000 0001	
r	0000 0000	
a&t	0000 0001	
r+	0000 0001	=> $r = 1_{10}$
a>>1	0000 0110	
a&t	0000 0000	
r+	0000 0001	=> $r = 1_{10}$
a>>1	0000 0011	
a&t	0000 0001	
r+	0000 0010	=> $r = 2_{10}$
a>>1	0000 0001	
a&t	0000 0001	
r+	0000 0011	=> $r = 3_{10}$
a>>1	0000 0000 (no more ones, check 0)	

MEMORIAL UNIVERSITY
www.mun.ca

More Examples

- D2L notes have examples in python, java and c.
  - Try those yourself
- Single.java (Java)
- andbits.c ©
  - gcc -O1 andbits.c
- Assembly code
  - C to assembly
  - QEmu
  - arm-linux-gnueabi-gcc -static andbits.c
  - qemu-arm ./a.out 1100 1010

Boolean Operations in Programming Languages

- Most programming languages provide the boolean operators
  - AND (&), OR (|), XOR (^), and NOT (~).
- For a single bit, the operators produce:

```
0 & 0 is 0, 0 & 1 is 0, 1 & 0 is 0, 1 & 1 is 1
0 | 0 is 0, 0 | 1 is 1, 1 | 0 is 1, 1 | 1 is 1
0 ^ 0 is 0, 0 ^ 1 is 1, 1 ^ 0 is 1, 1 ^ 1 is 0
~0 is 1, ~1 is 0
```

Bitwise Operators in Python

- Single bit boolean
- Bitwise operations
- 8-bit

Shifting

- Most programming languages support shifting the bits in the integer variables
- The bits can be shifted left or right
- Right shifting
  - either logic right shift (zeros are shifted in)
  - arithmetic right shift (the sign bit is retained, and its value is copied when shifted).

Shifting

The shifting operators are:

a << b   // shift the bits in **a** to the left by **b** bits

a >> b   // shift the bits in **a** to the right by **b** bits

a >>> b   // logical right shift in Java

MEMORIAL UNIVERSITY
www.mun.ca

Shifting

Python:

```
print( bin( 0b0110 << 3 ) )

print( bin( 0b0110000 >> 2 ) )

print( bin( 0b0110000 >> 0 ) )
```

Shifting

Shifting left by n, is the same as multiplying by 2**n. For example:

```
print( 3 << 3 )

print( 3 * 8 )

print( 10 << 1 )

print( 10 * 2 )

print( 5 << 5 )

print( 5 * 32 )
```

Shifting

Shifting right by n, is the same as dividing by 2n. For example:

```
print( 90 >> 1 )

print( 90 // 2 )   # integer division in python3

print( 90 >> 3 )

print( 90 // 8 )

print( -40 >> 1 )  # "sign bit" is copied

print( -40 // 2 )
```

Extracting Bits

A combination of boolean operators and shifting can be used to extract bits from an integer.

```
a = 0b1101101  # a 7 bit integer, bit 6 to bit 0

print( bin(a) )

t = (a & 0b111)   # extracting lower three bits

print( bin(t) )

t = ((a >> 2) & 0b111)   # extracting bits 4,3,2

print( bin(t) )

t = ((a >> 6) & 0b1)   # extracting bits 6

print( bin(t) )
```

Replacing Bits

A combination of boolean operators and shifting can also be used to replace bits in an integer.

```
a = 0b1101001  # a 7 bit integer, bit 6 to bit 0

print( bin(a) )

t = ((a & ~0b111) | 0b110)   # replace lower three bits with 0b110

print( t )

t = ((a & ~0b11100) | 0b10100)   # replace bits 4,3,2

print( bin(t) )
```


MEMORIAL UNIVERSITY
www.mun.ca

Questions?

Next: Verilog for combinational logic