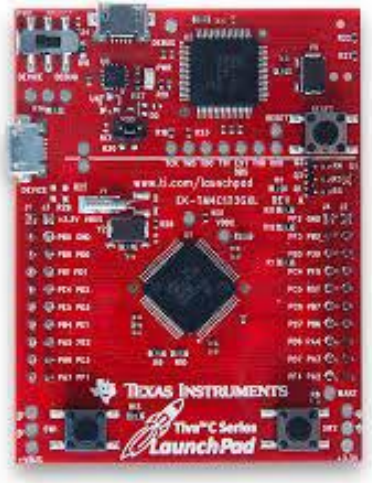
SDU

Bit Manipulation



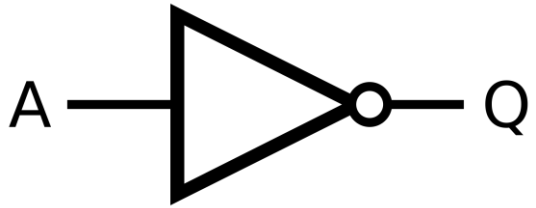
Logic functions

- The four axioms:
  - A variable X can be 0 or 1.
    If it is 0, it can not be 1, and if it is 1, it can not be 0.
  - The NOT function
  - The OR function
  - The AND function

SDU

The NOT function

- $Q = \bar{A}$

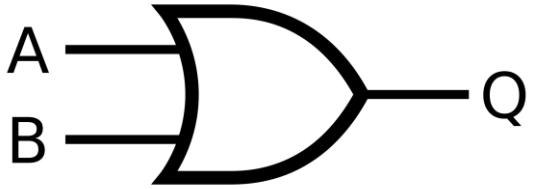


A	Q
0	1
1	0

SDU

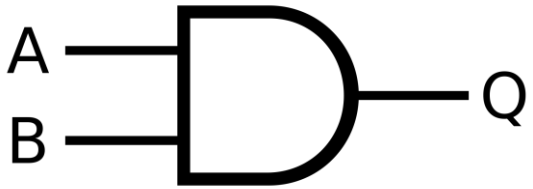
The OR and AND functions

- OR $Q = A + B$



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

- AND $Q = A \cdot B$

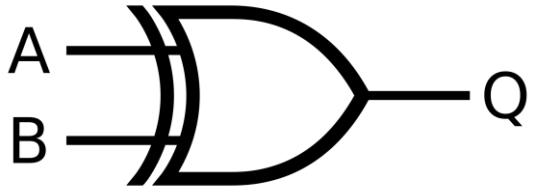


A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

SDU

The XOR function

- $Q = A \oplus B$
- Exclusive or
- Useful for bitwise comparison and toggling bits

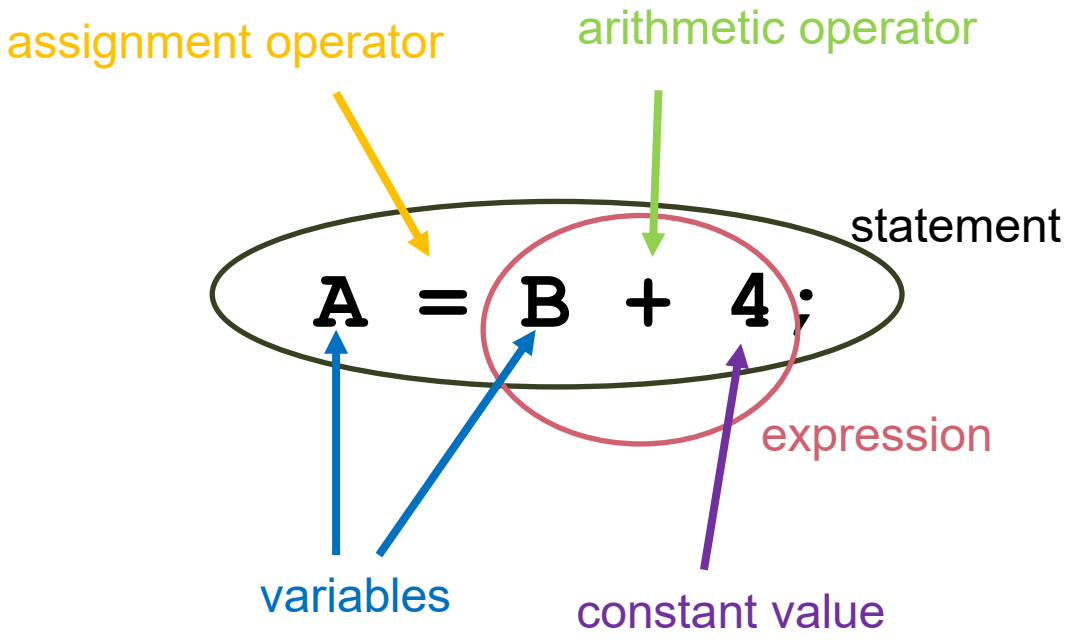


A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

SDU

Understanding a C assignment statement

- The variable on the left side of the equation will take the value of the expression on the right side



SDU

Operators in C

- Operators are symbols which tell the compiler to perform certain operations on variables and constants
- Types:
  - Arithmetic: +, -, *, /, %, e.g. a+3, b/c
  - Increment and decrement operators: ++, --, e.g. a++, ++a, a--
  - Assignment and compound: =, +=, -=, /=, e.g. a = b + 3, m+=43
  - Relational: ==, >, <, !=, >=, <=, e.g. if(a==3)
  - Logical: &&, ||, !
  - Bitwise: &, |, ^, ~, <<, >>

SDU

Logical vs bitwise operators in C

- Logical operators operate on bytes and words
  - An expression is true if it is different from 0
  - An expression is false if it is equal to 0
  - The outcome of these operations are 0 or 1

- Bitwise operators perform logical functions by comparing bits one by one

Logical NOT	!	!a
Logical AND	&&	a && b
Logical OR	||	a || b

Bitwise NOT	~	~a	A= ~(0x55);
Bitwise AND	&	a & b	if(B & 0x10)
Bitwise OR	|	a | b	A |=0x0F;
Bitwise XOR	^	a ^ b	PORT ^= 0x01
Bitwise left shift	<<	a << b	A = B << 4;
Bitwise right shift	>>	a >> b	A = B >> 2;

SDU

Compound bitwise operators in C

- Read modify write in C

```
Temp1 = Port;
Temp2 = Temp1 & 0xFFF7; // Clear bit 3
Port = Temp2;
```

```
Port &= 0xFFF7; // Clear bit 3
```


SDU

Compound bitwise operators in C

- Combines assignment with bitwise operation

Bitwise NOT	~	~a
Bitwise AND	&	a & b
Bitwise OR	|	a | b
Bitwise XOR	^	a ^ b
Bitwise left shift	<<	a << b
Bitwise right shift	>>	a >> b

Compound		
Bitwise AND	a &= b	a = a & b
Bitwise OR	a |= b	a = a| b
Bitwise XOR	a ^= b	a = a ^ b
Bitwise left shift	a <<= b	a = a << b
Bitwise right shift	A >>= b	a = a >> b

SDU

Numeric notation in C

- Hexadecimal notation
  - 0x10 = 16
  - 0xff = 255

Decimal	Hexadecimal	Binary
0	0x00	0000
1	0x01	0001
2	0x02	0010
3	0x03	0011
4	0x04	0100
5	0x05	0101
6	0x06	0110
7	0x07	0111
8	0x08	1000
9	0x09	1001
10	0x0a	1010
11	0x0b	1011
12	0x0c	1100
13	0x0d	1101
14	0x0e	1110
15	0x0f	1111

SDU

The equal sign in C

- ‘=‘ assigns a value to a variable
  - E.g. `Var = 6;  // assign the value 6 to Var`
- ‘==‘ compares two values
  - E.g. `if (Var == 6)  // ask if the value of Var is 6`
- Many bugs may come out of this
  - E.g. `if (Var = 6)`

SDU

Control individual bit(s)

- Set a bit
  - `Var |= 0x20;  // set bit 5`
- Clear a bit
  - `Var &= 0xFF9F; // clear bit 5 and 6`
  - Or
  - `Var &= ~(0x60)`
- Toggle a bit
  - `Var ^= 0x02;  // toggle bit 1`

Decimal	Hexadecimal	Binary
0	0x00	0000
1	0x01	0001
2	0x02	0010
3	0x03	0011
4	0x04	0100
5	0x05	0101
6	0x06	0110
7	0x07	0111
8	0x08	1000
9	0x09	1001
10	0x0a	1010
11	0x0b	1011
12	0x0c	1100
13	0x0d	1101
14	0x0e	1110
15	0x0f	1111

Example

```
Var = 0xf7
```

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
1	1	1	1	0	1	1	1

```
Var &= ~(0x60)
```

```
0x60
```

0	1	1	0	0	0	0	0

```
~(0x60)
```

1	0	0	1	1	1	1	1

```
0xf7 & ~(0x60)
```

1	1	1	1	0	1	1	1
1	0	0	1	1	1	1	1
1	0	0	1	0	1	1	1

```
0x97
```

Dec	Hex	Bin
0	0x00	0000
1	0x01	0001
2	0x02	0010
3	0x03	0011
4	0x04	0100
5	0x05	0101
6	0x06	0110
7	0x07	0111
8	0x08	1000
9	0x09	1001
10	0x0a	1010
11	0x0b	1011
12	0x0c	1100
13	0x0d	1101
14	0x0e	1110
15	0x0f	1111

Test individual bit(s)

- Test if bit is set
  - `if(Var & 0x04);   // if bit 2 is set`

Decimal	Hexadecimal	Binary
0	0x00	0000
1	0x01	0001
2	0x02	0010
3	0x03	0011
4	0x04	0100
5	0x05	0101
6	0x06	0110
7	0x07	0111
8	0x08	1000
9	0x09	1001
10	0x0a	1010
11	0x0b	1011
12	0x0c	1100
13	0x0d	1101
14	0x0e	1110
15	0x0f	1111