E91
Embedded System

Intro

Why embedded systems?

- Big bang-for-the-buck by adding some intelligence to systems.
- Embedded Systems are ubiquitous.
- Embedded Systems more common as prices drop, and power decreases.

Which Embedded System?

- We will use Texas Instruments MSP-430
  - + very popular (#2 next to microchip)
  - + 16 bits (instead of 8)
  - + low power
  - + clean architecture
  - + low cost (free) development tools
  - - relatively low speed/capacity (i.e., no video or fancy audio)
  - - low level tools (compared to Stamp, Arduino...)
  - - 16 bits (instead of 32)

Course overview

- Mix of hardware and software
- Mostly C, some assembly
- Lots of information, less conceptual than many courses.
- Two hardware kits, EZ430 and the 430 Experimenter's Board
- About 2 hours of lecture per week.
- More hours in lab – writeups will be short.
- In lab you will need to become familiar with a broad set of documentation. I will try to keep a complete list on website. Let me know if you find good documents I have not listed.
- You will need to be more self-reliant than in many courses (though I am always ready to help).
- Maybe some assignments.

Policy on working together :

- Lab groups should be 2 (preferably) or 3 students.
- We expect labs to be done as a group with your lab partners. You may discuss your lab with other groups, but you may not copy anything from their reports. Each group will submit a single report (with all members of the group listed).

Today

- Brief overview of
  - logic
  - numbers
  - C
  - MSP430 digital I/O

Today's lecture will be very densely packed so you can start on the lab. We will come back and repeat some of the material in more depth as needed.

Brief Review of Logic

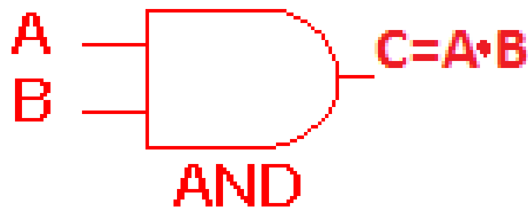
- In this class we will deal with logical circuits (i.e., inputs are either true (logical 1, for us this is 3.3V) or false (logical 0, for us this is 0V). In class you will learn why this is useful, for now, accept it.
- To deal with these signal levels we develop a special form of mathematics, Boolean algebra.

Boolean Operators

- Operators:
  - AND : $C=A\cdot B$ (Read as C equals A and B).

Input A	Input B	Output C
0	0	0
0	1	0
1	0	0
1	1	1

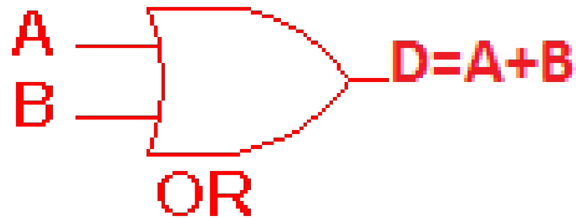
"Truth Table"



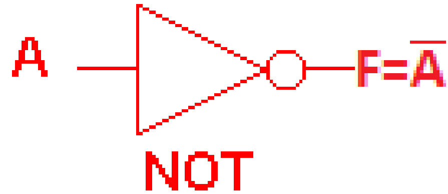
Sometimes the "·" is dropped: C=AB

logic gate images adapted from: http://www.ee.surrey.ac.uk/Projects/Labview/gatesfunc/index.html

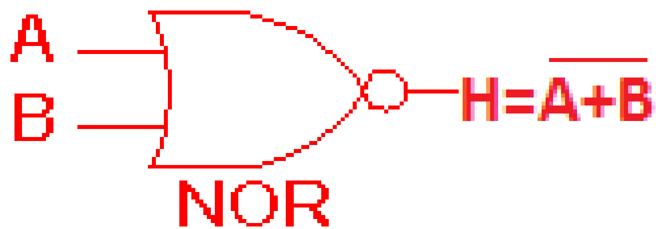
More logic…



A	B	D
0	0	0
0	1	1
1	0	1
1	1	1

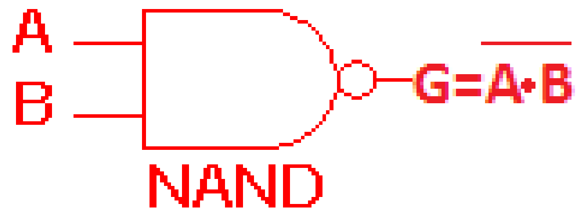


A	F
0	1
1	0

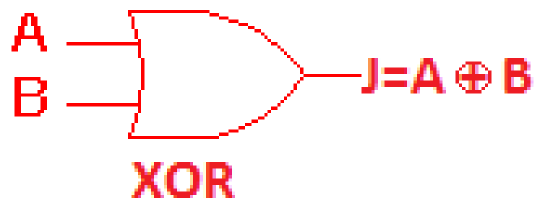


A	B	H
0	0	1
0	1	0
1	0	0
1	1	0

… and even more.



A	B	G
0	0	1
0	1	1
1	0	1
1	1	0



eXclusive OR

A	B	J
0	0	0
0	1	1
1	0	1
1	1	0

Note:
XOR with 1 inverts bit,
XOR with 0 passes bit.

Number Systems

Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Binary: $00001101_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 1 = 13$

Hex: $00101010_2 = 2A_{16} = 0x2A = 2 \cdot 16^1 + 10 \cdot 16^0 = 32 + 10 = 42$

(check $1 \cdot 2^5 + 1 \cdot 2^3 + 2 \cdot 2^1 = 32 + 8 + 2 = 42$)

8 bits = 1 byte

$0000\ 0000_2 \rightarrow 1111\ 1111_2$

$0x00 \rightarrow 0xff$

$0 \rightarrow 2^8\text{-}1 = 255 \ \left(\text{or } -128 \rightarrow 127 \ \left(-(2^7) \rightarrow 2^7\text{-}1\right)\right)$

16 bits = 2 bytes = 1 word

$0000\ 0000\ 0000\ 0000_2 \rightarrow 1111\ 1111\ 1111\ 1111_2$

$0x0000 \rightarrow 0xffff$

$0 \rightarrow 2^{16}\text{-}1 = 65535 \ \left(\text{or } -32768 \rightarrow 32767 \ \left(-(2^{15}) \rightarrow 2^{15}\text{-}1\right)\right)$

4 bits = 1 nybble $(0 \rightarrow 2^4\text{-}1 = 15)$

A simple C program

```
#include  <msp430x20x3.h>

void main(void) {
volatile int i;
  WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
  P1DIR |= 0x01;            // Set P1.0 to output direction

  while (1) {  //Do this forever
       P1OUT = P1OUT | 0x01;       // Set P1.0 with "or", |
       for (i=0; i<0x5000; i++) {}  // Delay
       P1OUT = P1OUT & ~0x01;      // Clear P1.0
       for (i=0; i<0x5000; i++) {}  // Delay
  }
}
```

Constants associated with our chip

Every program needs a "main" routine (between braces)

Declare "i" as volatile so compiler doesn't optimize it out of existence (or turn optimizations off). **All** variables must be declare before they are used.

"1" is always true, so loop forever.

Set bit 0 high (connected to LED)

Loop to waste time

Set bit 0 low (LED turns off)

Loop to waste time

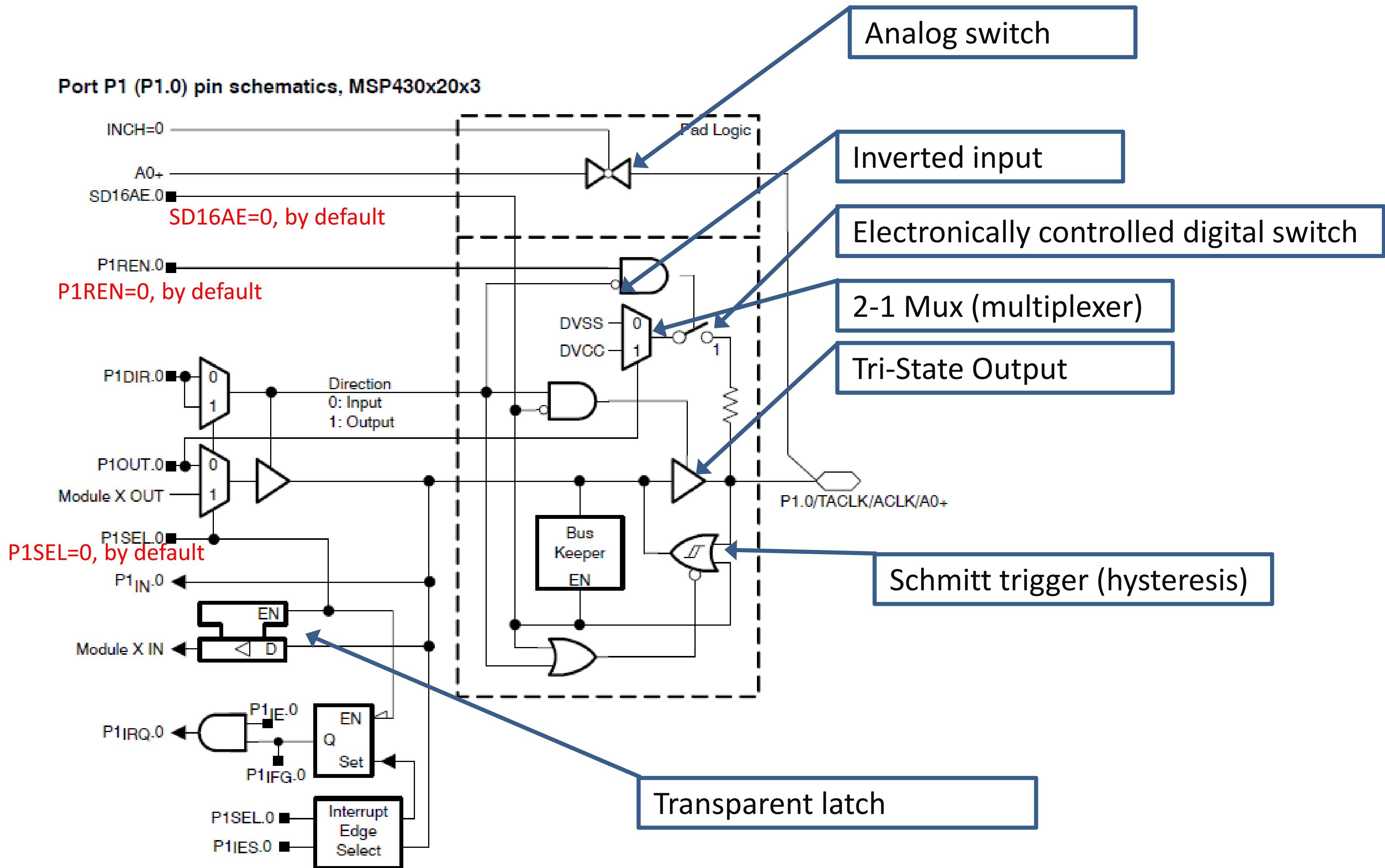
Don't worry about for now.

Set bit 0 in "P1DIR" - this makes it an output (next page).

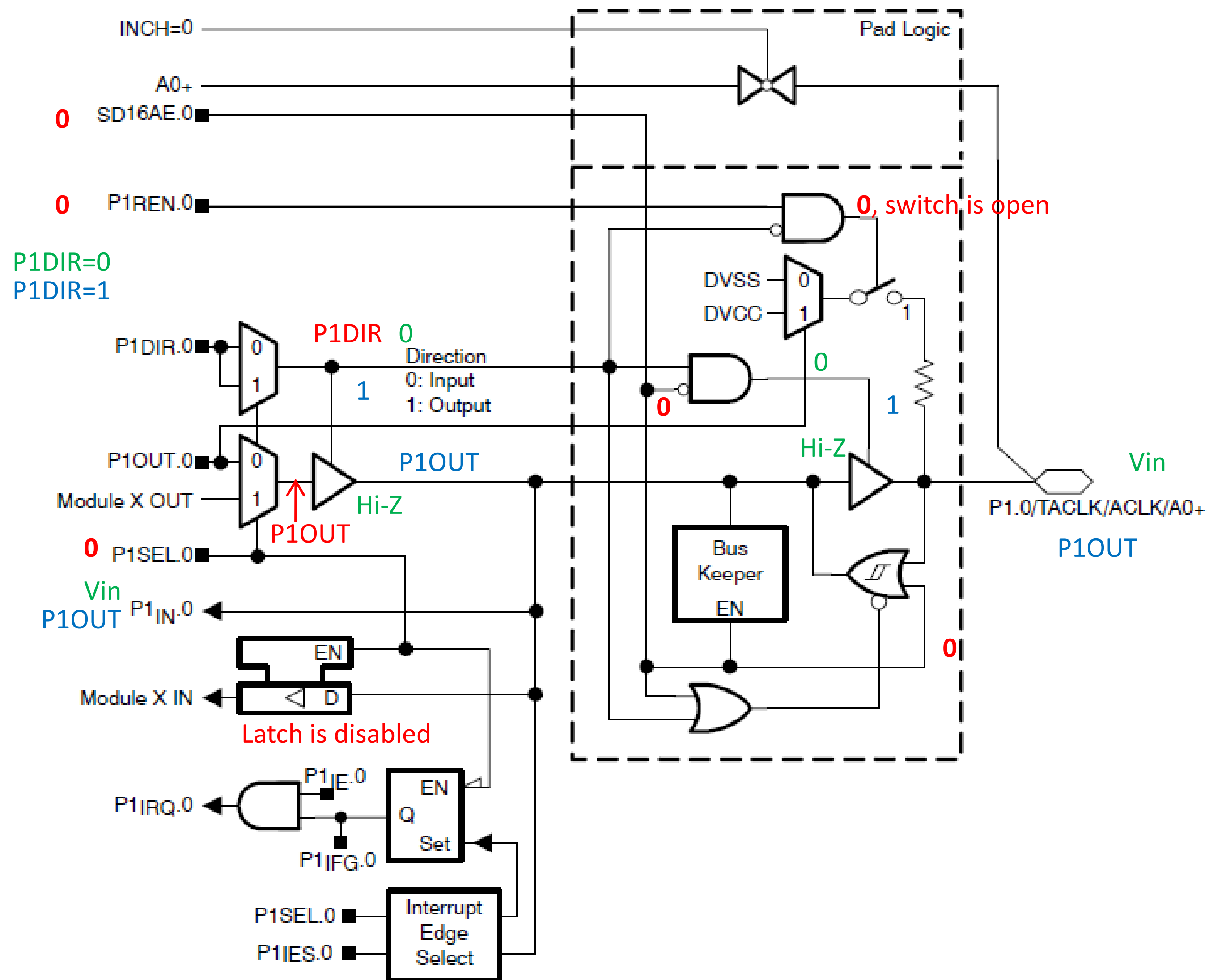
Comments start with "//" and go to end of line.

Also note that every statement ends with ";" or "}"

A typical I/O pin



Effect of P1DIR



Variant 1 (more readable)

```
#include  <msp430x20x3.h>
#define LED    0x01

void main(void) {
volatile int i;
  WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
  P1DIR |= LED;             // Set P1.0 (LED bit) to output

  while (1) {  //Do this forever
        P1OUT |= LED;                   // Turn on LED
        for (i=0; i<0x5000; i++) {}     // Delay
        P1OUT &= ~0x01;                 // Turn off LED
        for (i=0; i<0x5000; i++) {}     // Delay
  }
}
```

Give constants meaningful names.

Equivalent to: `P1OUT = P1OUT | LED;`

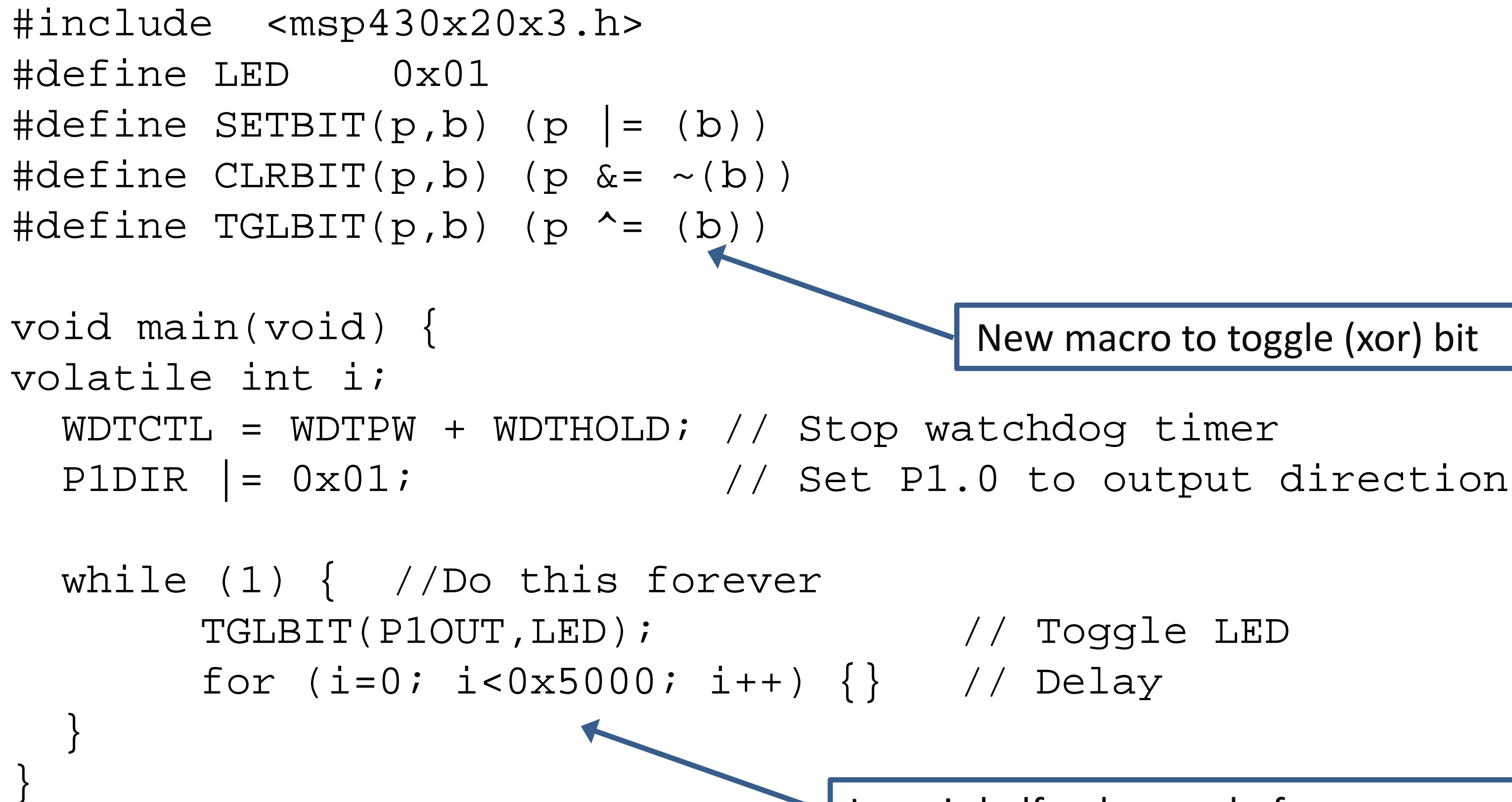
Variant 2 (macros)

```
#include  <msp430x20x3.h>
#define LED     0x01
#define SETBIT(p,b) (p |= (b))
#define CLRBIT(p,b) (p &= ~(b))

void main(void) {
volatile int i;
  WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
  P1DIR |= LED;               // Set P1.0 to output direction

 while (1) {  //Do this forever
       SETBIT(P1OUT,LED);              // Set P1.0
       for (i=0; i<0x5000; i++) {}   // Delay
       CLRBIT(P1OUT,LED);              // Clear P1.0
       for (i=0; i<0x5000; i++) {}   // Delay  }
}
```

Use Macros sparingly, but they can make code look much cleaner (see below)

Expands to: `(P1OUT |= (0x01))`
Note ";" must be added.

Variant 3 (shorter)




```
#include  <msp430x20x3.h>
#define LED    0x01
#define SETBIT(p,b) (p |= (b))
#define CLRBIT(p,b) (p &= ~(b))
#define TGLBIT(p,b) (p ^= (b))

void main(void) {
volatile int i;
  WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
  P1DIR |= 0x01;            // Set P1.0 to output direction

  while (1) {  //Do this forever
        TGLBIT(P1OUT,LED);            // Toggle LED
        for (i=0; i<0x5000; i++) {}   // Delay
  }
}
```

C Data Types
(that we will use)

Type	Size (bits)	Representation	Minimum	Maximum
char, signed char	8	ASCII	-128	+127
unsigned char, bool	8	ASCII	0	255
int, signed int	16	2s complement	-32 768	32 767
unsigned int	16	Binary	0	65 535
long, signed long	32	2s complement	-2 147 483 648	2 147 483 647
unsigned long	32	Binary	0	4 294 967 295
enum	16	2s complement	-32 768	32 767
float	32	IEEE 32-bit	±1.175 495e-38	±3.40 282 35e+38

C Operators
(Arithmetic)

Arithmetic Operator name		Syntax
Basic assignment		a = b
Addition		a + b
Subtraction		a - b
Unary plus		+a
Unary minus (additive inverse)		-a
Multiplication		a * b
Division		a / b
Modulo (remainder)		a % b
Increment	Prefix	++a
	Suffix	a++
Decrement	Prefix	--a
	Suffix	a--

More C Operators

(Relational, Logical, Bitwise and Compound)

Relational Operator name	Syntax
Equal to	a ==b
Not equal to	a !=b
Greater than	a > b
Less than	a < b
Greater than or equal to	a >=b
Less than or equal to	a <=b

Bitwise Operator name	Syntax
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

Logical Operator name	Syntax
Logical negation (NOT)	!a
Logical AND	a &&b
Logical OR	a ||b

Compound Operator name	Syntax
Addition assignment	a += b
Subtraction assignment	a -= b
Multiplication assignment	a *= b
Division assignment	a /= b
Modulo assignment	a %= b
Bitwise AND assignment	a &= b
Bitwise OR assignment	a |= b
Bitwise XOR assignment	a ^= b
Bitwise left shift assignment	a <<=b
Bitwise right shift assignment	a >>=b

More C

Statements

- a **simple statement** is a single statement that ends in a ";"
- a **compound statement** is several statements inside braces:

```
{
simple statement;
 …
 simple statement;
}
```

Indenting

There are no rules about indenting code, but if you don't adopt a standard style, your code becomes unreadable.

```
while (x == y) {
    something();
    somethingelse();
    if (some_error)
        do_correct();
    else
        continue_as_usual();
}
```

```
while (x == y)
{
    something();
    somethingelse();
}
finalthing();
```

```
if (x < 0)
{   printf("Negative");
    negative(x);
}
else
{   printf("Positive");
    positive(x);
}
```

Even more C

Array definition

```
int a [100];    //Array elements are a[0] to a[99].  Don’t use a[100]!
```

if…then

```
if (<expression>)
        <statement>
```

<statement> may be a compound statement.

if…then…else

```
if (<expression>)
        <statement1>
else
        <statement2>
```

Yet more C

Iteration (do…while while… for…)

```
do
        <statement>
while ( <expression> );
```

```
while ( <expression> )
        <statement>
```

```
for ( <expression> ; <expression> ; <expression> )
        <statement>
```

Recall: `for (i=0; i<0x5000; i++) {}    // Delay`

```
for (e1; e2; e3)
        s;
```

is equivalent to

```
e1;
while (e2) {
        s;
        e3;
}
```

The break statement is used to end a for loop, while loop, do loop, or switch statement. Control passes to the statement following the terminated statement.

Again with the C

switch (one choice of many)

```
switch (<expression>) {
        case <label1> :
                <statements 1>
        case <label2> :
                <statements 2>
                break;
        default :
                <statements 3>
}
```

- <expression> is compared against the label, and execution of the associated statements occur (i.e., if <expression> is equal to <label1>, <statements 1> are exectuted.
- No two of the case constants may have the same value.
- There may be at most one default label.
- If none of the case labels are equal to the expression in the parentheses following switch, control passes to the default label, or if there is no default label, execution resumes just beyond the entire construct.
- Switch statements can "fall through", that is, when one case section has completed its execution, statements will continue to be executed downward until a break; statement is encountered. This is usually not wanted, so be careful

Material taken from:

- http://en.wikipedia.org/wiki/C_syntax
- http://en.wikipedia.org/wiki/Indent_style
- http://en.wikipedia.org/wiki/Operators_in_C_and_C++
- http://focus.ti.com/lit/ug/slau144e/slau144e.pdf
- http://focus.ti.com/lit/ds/slas491e/slas491e.pdf
- http://focus.ti.com/lit/ug/slau132c/slau132c.pdf