Data type declaration *	Number of bits	Range of values
char k; unsigned char k; uint8_t k;	8	0..255
signed char k; int8_t k;	8	-128..+127
short k; signed short k; int16_t k;	16	-32768..+32767
unsigned short k; uint16_t k;	16	0..65535
intk; signed intk; int32_t k;	32	-2147483648.. +2147483647
unsigned intk; uint32_t k;	32	0..4294967295

intx_tand uintx_tdefined in stdint.h

Automatic variable example.

- Declare within a function/procedure
- Variable is visible (has scope) only within that function
  - Space for the variable is allocated on the system stackwhen the procedure is enteredDeallocated, to be re-used, when the procedure is exited
  - If only 1 or 2 variables, the compiler may allocate them to registerswithin that procedure, instead of allocating memory.
  - Values are not retained between procedure calls

Variables must be initialized each time the procedure is entered since values are not retained when the procedure is exited.

```
void delay ()
{
        inti,j; //automatic variables –visible only within delay()
        for (i=0; i<100; i++)
        { //outer loop
                for (j=0; j<20000; j++)
                { //inner loop
                } //do nothing
        }
}
```

Static variable example

```
unsigned char count; //global variable is static –allocated a fixed RAM location
//count canreferenced by any function

void math_op() {
        inti;//automatic variable –allocated space on stack when function entered
        static intj; //static variable –allocated a fixed RAM location to maintain the value
        if (count == 0) //test value of global variable count
```

```
            j = 0;//initialize static variable j first time math_op() entered
        i= count;//initialize automatic variable ieach time math_op() entered
        j = j + i; //change static variable j –value kept for next function call
}//return & deallocatespace used by automatic variable i

void main(void) {
        count = 0;//initialize global variable count
        while (1) {
        math_op();
        count++;//increment global variable count
        }
}
```

Arithmetic operations

C examples –with standard arithmetic operators

```
inti, j, k;             // 32-bit signed integers
uint8_t m,n,p;          // 8-bit unsigned numbers
i= j + k;               // add 32-bit integers
m = n -5;               // subtract 8-bit numbers
j = i* k;               // multiply 32-bit integers
m = n / p;              // quotient of 8-bit divide
m = n % p;              // remainder of 8-bit divide
i= (j + k) * (i–2);     //arithmetic expression
```

*, /, % are higher in precedence than +, -(higher precedence applied 1st)
Example: j * k + m / n = (j * k) + (m / n)

Bit-parallel logical operators

Bit-parallel (bitwise) logical operators produce n-bit results of the corresponding logical operation: &(AND) |(OR) ^(XOR) ~(Complement)

```
C = A & B;A0 1 1 0 0 1 1 0
      (AND)B1 0 1 1 0 0 1 1
              C0 0 1 0 0 0 1 0

  C = A | B;A0 1 1 0 0 1 0 0
        (OR)B0 0 0 1 0 0 0 0
              C0 1 1 1 0 1 0 0

C = A ^ B;A0 1 1 0 0 1 0 0
     (XOR)B1 0 1 1 0 0 1 1
            C1 1 0 1 0 1 1 1
B = ~A;              A0 1 1 0 0 1 0 0
(COMPLEMENT)B1 0 0 1 1 0 1 1
```

Bit set/reset/complement/test

Use a “mask” to select bit(s) to be altered

```
C = A & 0xFE;A  a b c d e f g h Clear selected bit of A
              0xFE 1 1 1 1 1 1 1 0
```

C a b c d e f g 0

C = A & 0x01;A a b c d e f g h Clear all but the selected bit of A
0xFE 00 0 0 0 0 0 1
C 0 0 0 0 0 0 0 h

C = A | 0x01;A a b c d e f g h Set selected bit of A
0x01 0 0 0 0 0 0 0 1
C a b c d e f g 1

C = A ^ 0x01;A a b c d e f g h Complement selected bit of A
0x01 0 0 0 0 0 0 0 1
C a b c d e f g h'

Bit examples for input/output

- Create a "pulse" on bit 0 of PORTA (assume bit is initially 0)

  PORTA = PORTA | 0x01; //Force bit 0 to 1
  PORTA = PORTA & 0xFE; //Force bit 0 to 0
- Examples:
  if ((PORTA & 0x80) != 0) //Or: ((PORTA & 0x80) == 0x80)
  bob(); // call bob() if bit 7 of PORTA is 1
  c = PORTB & 0x04; // mask all but bit 2of PORTB value
  if ((PORTA & 0x01) == 0) // test bit 0 of PORTA
  PORTA = c | 0x01; // write c to PORTA with bit 0 set to 1

Shift operators

Shift operators:
x >> y (right shift operand x by y bit positions)
x << y (left shift operand x by y bit positions)
Vacated bits are filled with 0's.
Shift right/left fast way to multiply/divideby power of 2

B = A << 3; A1 0 1 0 1 1 0 1
(Left shift 3 bits) B0 1 1 0 1 0 0 0

B = A >> 2; A1 0 1 1 0 1 0 1
(Right shift 2 bits) B0 0 1 0 1 1 0 1

B = '1'; B =0 0 1 1 0 0 0 1 (ASCII 0x31)
C = '5'; C = 0 0 1 1 0 1 0 1(ASCII 0x35)

D = (B << 4) | (C & 0x0F);
(B << 4) = 0 0 0 10 0 0 0
(C & 0x0F)= 0 0 0 0 0 1 0 1
D = 0 0 0 1 0 1 0 1 (Packed BCD 0x15)

Relational Operators

Test relationship between two variables/expressions

Test	TRUEcondition	Notes

(m == b)	mequal to b	Double =
(m!= b)	m not equal to b	
(m < b)	m less than b	1
(m <= b)	m less than or equal to b	1
(m > b)	mgreater than b	1
(m >= b)	m greater than or equal to b	1
(m)	mnon-zero	
(1)	always TRUE	
(0)	always FALSE	

Boolean operators

- Boolean operators &&(AND) and ||(OR) produce TRUE/FALSE results when testing multiple TRUE/FALSE conditions

  if ((n > 1) && (n < 5)) //test for n between 1 and 5
  if ((c = 'q') || (c = 'Q')) //test c = lower or upper case Q

- Note the difference between Booleanoperators &&, || and bitwise logical operators &, |

  if (k && m) //test if k and m both TRUE (non-zero values)
  if (k & m) //compute bitwise AND between m and n,
  //thentest whether the result is non-zero (TRUE)

Function arguments

Calling program can pass information to a function in two ways

- By value: pass a constant or a variable valuefunction can use, but not modify the value
- By reference: pass the address of the variablefunction can both read and update the variable
- Values/addresses are typically passed to the function by pushing them onto the system stackFunction retrieves the information from the stack

Example –pass by reference

```
/* Function to calculate x2*/
void square ( intx, int*y) { //value of x, address of y
        *y = x * x;//write result to location whose address is y
}

void main {
intk,n;//local variables –scope limited to main
n = 5;
        square(n, &k);//calculate n-squared and put result in k
        square(5, &n); // calculate 5-squared and put result in n
}
```

In the above, *main*tells *square*the location of its local variable, so that *square*can write the result to that variable.