Sunbeam Institute of Information Technology
Pune and Karad

Module - Embedded C Programming

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Print binary

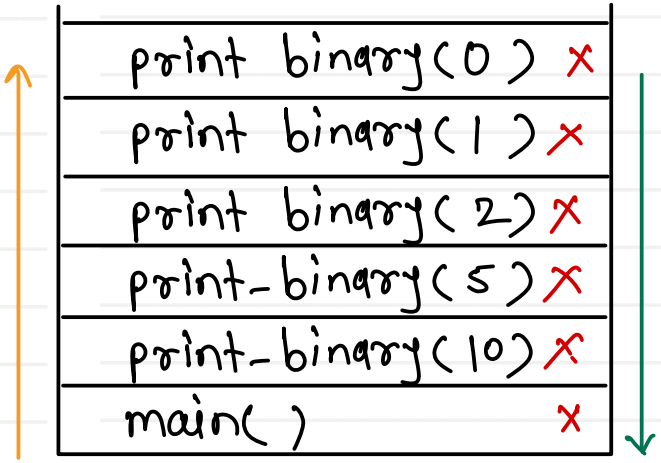
2	10	
	5	0
	2	1
	1	0
		1

$(10)_{10} = (1010)_2$

```
void print_binary(int num) {
    while (num > 0) {
        printf("%d", num % 2);
        num /= 2;
    }
}
```

num	num>0	num % 2
10	T	0
5	T	1
2	T	0
1	T	1
0	F	

```
void print_binary(int num) {
    if (num == 0)
        return;
    print_binary(num / 2);
    printf("%d", num % 2);
}
```



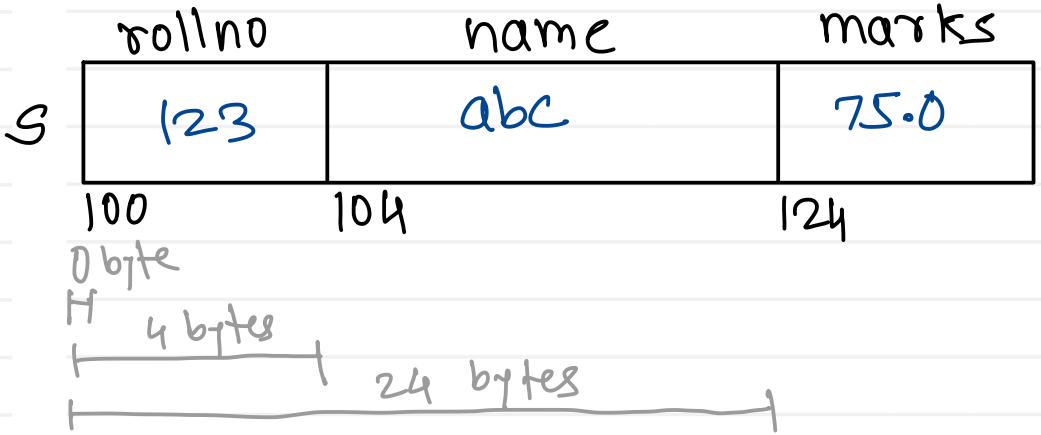
$(10)_2 = (1010)_2$

Structure members offset

Offset : displacement of a member from starting address of structure variable

```
struct student {
    int rollno;
    char name[20];
    float marks;
};
```

```
struct student s = { 123, "abc", 75.0f };
```



offset_of(member) =
address_of(member) – address_of(struct)

offset(rollno) = &rollno – &s = 100-100 = 0
offset(name) = &name – &s = 104-100 = 4
offset(marks) = &marks – &s = 124-100 = 24

Endianness

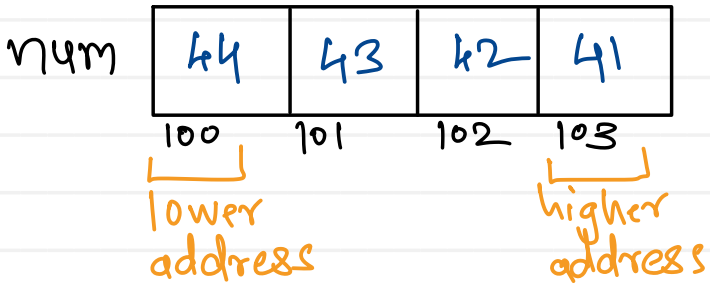
- it decides byte ordering in memory
- how multi byte variables will be stored in memory.

Little Endian

lower byte – lower address
higher byte – higher address

int num = 0x 41 42 43 44

44 – lower byte
41 – higher byte



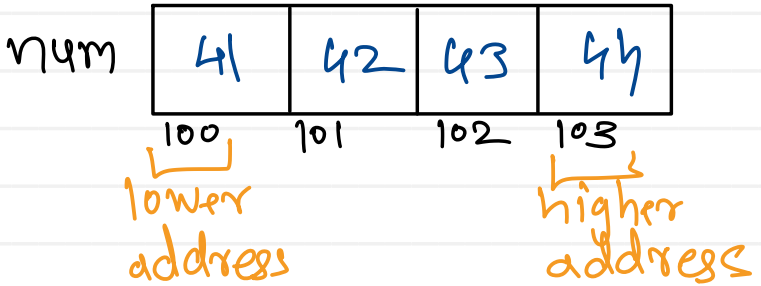
e.g. x86 / x86_64, ARM, AVR

Big Endian

lower byte – higher address
higherbyte – lower address

int num = 0x 41 42 43 44

41 – higher byte
44 – lower byte



e.g. powerPC, Networks

Function calling conventions

- Function calling conventions decides
  1. How arguments are pushed on stack
     left to right or right to left
  2. Who will do stack cleanup.
     caller (calling function)
     callee (called function)
  3. How CPU registers will be used

```
int fun(int n1, int n2, int n3)  <- callee
{
    //TO DO
}

int main(void)  <- caller
{
    fun(10,20,30);

    return 0;
}
```

1. Pascal - removed
2. cdecl - default for C/C++
3. stdcall
4. fastcall
5. thiscall - only for C++

Pascal	left to right	callee
cdecl	right to left	caller
stdcall	right to left	callee

cdecl

```
int fun( int n1 ,int n2, int n3)  <- callee
{
      //TO DO
}

int main (void)  <- caller
{
     fun(10,20,30);
      push(30)
      push(20)
      push(10)
       call
      pop()
      pop()
      pop()
   return 0;
}
```

stdcall

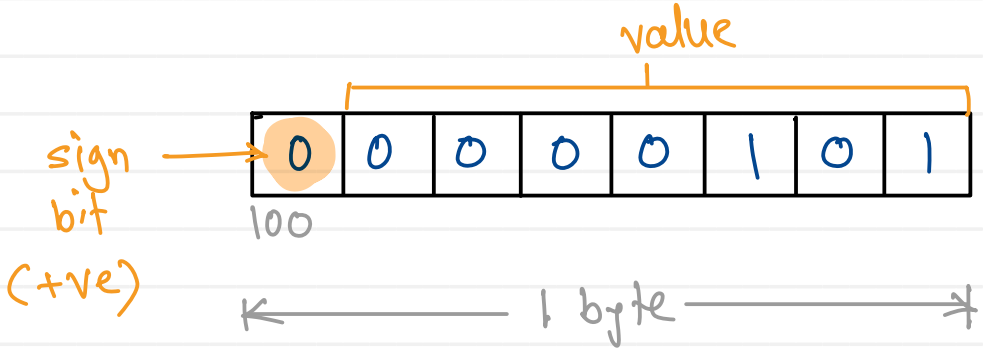
```
int fun( int n1 ,int n2, int n3)  <- callee
{
      //TO DO
      pop()
      pop()
      pop()
}
int main (void)  <- caller
{
    fun(10,20,30);
     push(30)
     push(20)
     push(10)
      call

    return 0;
}
```

signed variable

signed char num = 5;

5 = 0000 0101



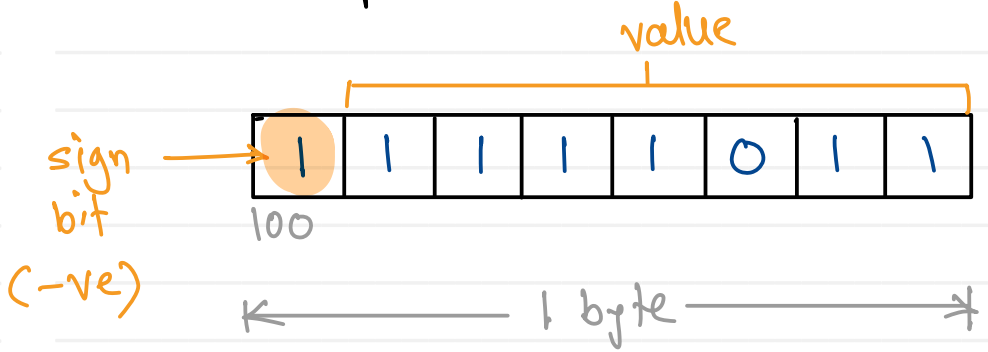
signed char num = -5;

5 = 0000 0101

1's complement = 1111 1010

+1 0000 0001

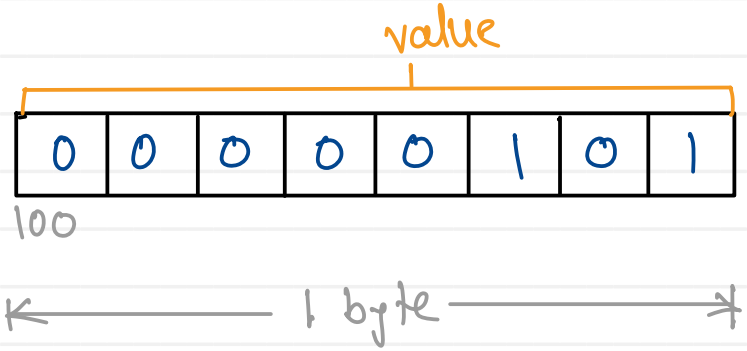
2's complement = 1111 1011



unsigned variable

unsigned char num = 5;

5 = 0000 0101



printf("%u", num)

output = 5

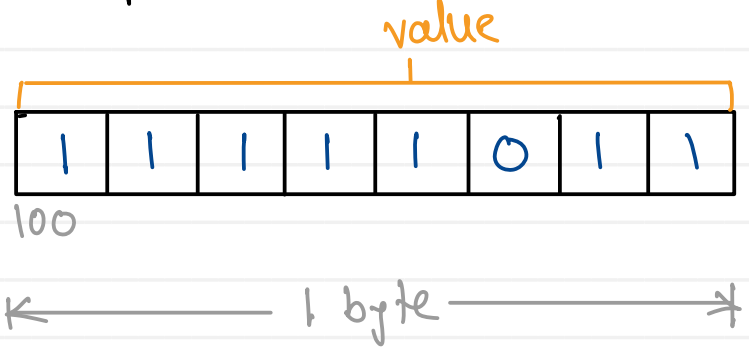
unsigned char num = -5;

5 = 0000 0101

1's complement = 1111 1010

+1 0000 0001

2's complement = 1111 1011



printf("%u", num)

output = 251

signed vs unsigned

```
int main(void) {
    signed char n1 = 5;
    signed char n2 = -5;

    if(n1 > n2)
        printf("n1 is greater");
    else
        printf("n2 is greater");

    return 0;
}
```

Output : n1 is greater

n1 = 0000 0101 = 5
n2 = 1111 1011 = -5

```
int main(void) {
    unsigned char n1 = 5;
    unsigned char n2 = -5;

    if(n1 > n2)
        printf("n1 is greater");
    else
        printf("n2 is greater");

    return 0;
}
```

Output : n2 is greater

n1 = 0000 0101 = 5
n2 = 1111 1011 = 251

Bit wise operators

Bitwise AND (&)

num1 : 0xAB : 1010 1011
num2 : 0x34 : 0011 0100
&
0010 0000 (0x20)

Bitwise XOR (^)

num1 : 0xAB : 1010 1011
num2 : 0x34 : 0011 0100
^
1001 1111 (0x9F)

Bitwise OR (|)

num1 : 0xAB : 1010 1011
num2 : 0x34 : 0011 0100
|
1011 1111 (0xBF)

Bitwise NOT (~) (Complement)

num1 : 0xAB : 1010 1011
~
0101 0100 (0x54)

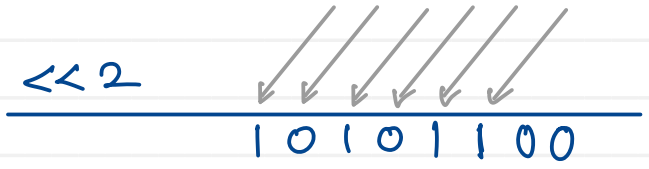
num2 : 0x34 : 0011 0100
~
1100 1011 (0xCB)

Bit wise operators

Bitwise Left shift (<<)

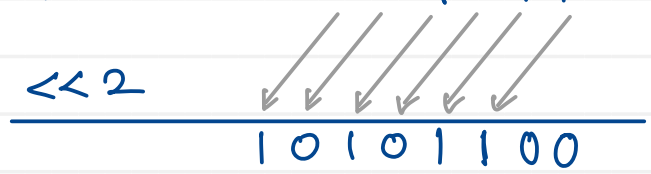
```
unsigned char num = 0xAB;
```

num : 0xAB : 1010 1011




```
signed char num = 0xAB;
```

num : 0xAB : 1010 1011

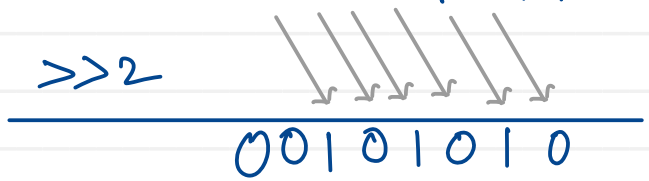


var << n ; bits will be shifted by n
0 will be added in n bits (vaccant) from LSB

Bitwise right shift (>>)

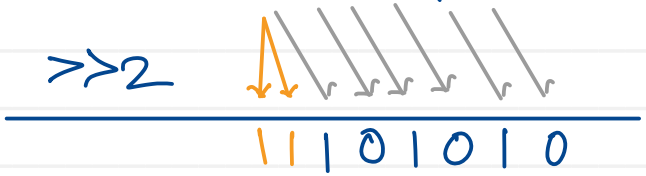
```
unsigned char num = 0xAB;
```

num : 0xAB : 1010 1011




```
signed char num = 0xAB;
```

num : 0xAB : 1010 1011



var >> n ; bits will be shifted by n
unsigned → 0 will be added in n bits from MSB
signed → MSB (sign bit) will be added in n bits from MSB

Bit wise operators

Even or Odd

num = 8 → 1000
= 9 → 1001
= 10 → 1010
= 11 → 1011

if LSB = 0
then num is even

if LSB = 1
then num is odd

0 & 1 = 0
1 & 1 = 1

```
num = 8;
if(num & 1)
    printf("num is odd");
else
    printf("num is even");
```

Divisible by 2

num = 8 → 1000
= 9 → 1001
= 10 → 1010
= 11 → 1011

if LSB = 0
then num is divisible

if LSB = 1
then num is not divisible

0 & 1 = 0
1 & 1 = 1

```
num = 8;
if(num & 1)
    printf("num is not divisible");
else
    printf("num is divisible");
```

Bit wise operators

Divisible by 4

num = 8 - 1000 9 - 1001
12 - 1100 11 - 1011
16 - 10000 15 - 1111
13 - 1101
14 - 1110

```
if( num & 3)
    printf("num is not divisible by 4");
else
    printf("num is divisible by 4");
```

Homework :
find lower & upper divisibk by 4 of num

num = 17
lower divisible = 16
Upper divisible = 20

num = 24
lower divisible = 24
upper divisible = 28

Swapping of numbers

n1 = 10 → 0000 1010
n2 = 20 → 0001 0100

n1 = n1 ^ n2 = 0000 1010
0001 0100
0001 1110

n2 = n1 ^ n2 = 0001 1110
0001 0100
0000 1010 (10)

n1 = n1 ^ n2 = 0001 1110
0000 1010
0001 0100 (20)

Print Collatz Conjecture

n >>1 → divide by 2

n = 0000 1000 → 8

>>1 = 0000 0100 → 4

n <<1 → multiply by 2

n= 0000 1000 → 8

<<1 = 0001 0000 → 16

n=8 multiply by 3

(n<<1) + n

```
n= 0000 1000
<<1 = 0001 0000
    + 0000 1000
-------------------
      0001 1000
```

collatz conjecture

num

if num is even → num/2

if num is odd → (num * 3) +1

num = 8

series = 8, 4, 2, 1

num = 13

series = 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

even/odd = num & 1

num /2 = num >>1

(num*3)+1 = num<<1 + num + 1

Print binary using bit wise operators

num = 8 = 0000 1000

```
& (0x80) 1000 0000 → 0
         0100 0000 → 0
         0010 0000 → 0
         0001 0000 → 0
         0000 1000 → nonzero → 1
         0000 0100 → 0
         0000 0010 → 0
         0000 0001 → 0
         0000 0000
```

```
char num = 8;
unsigned char mask = 0x80;

while(mask) {
    if(num & mask)
        printf("1");
    else
        printf("0");
    mask = mask >> 1;
}
```

Homework :

print binary of any size datatype variable

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com