Embedded C Programming

Agenda

- Bitwise operators
  - <<, >>
  - Applications
- Structures
  - Tagged initialization
  - Member offsets
  - . and -> operator internals
  - Slack bytes

Bitwise operators

- Bitwise AND
- Bitwise OR
- Bitwise NOT
- Bitwise XOR
- Precedance of Bitwise operators
  - preced(~) > preced(&) > preced(^) > preced(|)
  - 3 ^ 6 | 15 & ~5 = ((3 ^ 6) | (15 & (~5)))
- Left shift operator -- logical shift left (LSL)
  - num << n --> "n" bits are shifted to left.
    - "n" bits from MSB are discarded.
    - "n" 0 bits will be added on LSB.
  - 5 << 1 = 10

```
5 ->    0000 0101
<<              1
------------------
        0000 1010   --> 10
```

  - 5 << 2 = ??

```
5 ->    0000 0101
<<              2
------------------
        0001 0100   --> 20
```

    - 5 << n = 5 * 2$n
    - For an int (32-bit), num << 32 = 0
    - -5 << 2 = -20

- Right shift operator
    - (unsignend) num >> n --> "n" bits are shifted to right -- logical shift right (LSR)
        - "n" bits from LSB are discarded.
        - "n" 0 bits will be added on MSB.
    - (signend) num >> n --> "n" bits are shifted to right -- arithmetic shift right (ASR)
        - "n" bits from LSB are discarded.
        - "n" sign (+ve=0 or -ve=1) bits will be added on MSB.
    - 5 >> 1 = 2

```
5 ->    0000 0101
>>              1
------------------
        0000 0010   --> 2
```

    - 5 >> 2 = ???

```
5 ->    0000 0101
>>              2
------------------
        0000 0001   --> 1
```

    - 5 >> n = 5 / 2$n -- for +ve numbers and unsigned values.
    - (-5) >> 1 = 2

```
-5 ->   1111 1011
>>              1
------------------
        1111 1101   --> -3
```

```
-5 ->   1111 1011
>>              2
------------------
        1111 1110   --> -2
```

```
-5 ->   1111 1011
>>              4
```

```
------------------
        1111 1111   --> -1
```

Bitwise operators - Applications

- XOR operator: x ^ x = 0

```c
int main() {
    int num;
    printf("%d\n", num ^ num); // 01001011 ^ 01001011 = 00000000
    return 0;
}
```

- Swap two numbers without third variable.

```c
int main() {
    int a = 5, b = 9;
    a = a ^ b;  // a = 0101 ^ 1001 = 1100
    b = a ^ b;  // b = 1100 ^ 1001 = 0101 (5)
    a = a ^ b;  // a = 1100 ^ 0101 = 1001 (9)
    printf("a = %d, b = %d\n", a, b);   // a = 9, b = 5
    return 0;
}
```

- Check if given number is even or odd.

```c
int main() {
    int num;
    printf("enter a number: ");
    scanf("%d", &num);
    if(num % 2 != 0)    // % operator consume more CPU cycles
        printf("Odd\n");
    else
        printf("Even\n");
    return 0;
}
```

  - 6 --> 0110, 9 --> 1001, 4 --> 0100, 11 --> 1011
    - For even numbers LSB=0 and For odd numbers LSB=1
    - x & 1 = x, x & 0 = 0

```
num     (6) 0110        (9) 1001        (4) 0100
& 1         0001            0001            0001
```

```
-------------------------------------------------
            0000            0001            0000
```

```c
int main() {
    int num;
    printf("enter a number: ");
    scanf("%d", &num);
    if(num & 1)         // & operator consume single CPU cycle (if result=1
cond is true i.e ODD)
        printf("Odd\n");
    else
        printf("Even\n");
    return 0;
}
```

- Collatz Conjure
    - if prev term is odd, term * 3 + 1; and if prev term is even, term / 2. The last term is always 1.
        - e.g. 5, 16, 8, 4, 2, 1
        - e.g. 3, 10, 5, 16, 8, 4, 2, 1
        - e.g. 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

```c
int main() {
    int num;
    printf("enter a number: ");
    scanf("%d", &num);
    while(num != 1) {
        printf("%d, ", num);
        if(num % 2 != 0)
            num = num * 3 + 1;
        else
            num = num / 2;
    }
    printf("%d.\n", num); // 1
    return 0;
}
```

```c
int main() {
    int num;
    printf("enter a number: ");
    scanf("%d", &num);
    while(num != 1) {
        printf("%d, ", num);
        if(num & 1)
            num = (num << 1) + num + 1;
        else
            num = num >> 1;
```

```
    }
    printf("%d.\n", num); // 1
    return 0;
}
```

- Check if given number is divisible by 4 or not.
    - 6 --> 0110, 9 --> 1001, 4 --> 0100, 11 --> 1011, 8 --> 1000, 12 --> 1100
    - If number is divisible by 4 last 2 bits are 00.

```
num     (6) 0110        (9) 1001        (4) 0100    (8) 1000
& 3         0011            0011            0011        0011
-------------------------------------------------------------
            0010            0001            0000        0000
```

```
int main() {
    int num;
    printf("enter a number: ");
    scanf("%d", &num);
    if(num & 3)     // if result is non-zero (i.e. condition is true), then num is not divisible by 4 [ (num & 3) != 0 ]
        printf("Not divisible by 4\n");
    else
        printf("Divisible by 4\n");
    return 0;
}
```

- Check if nth bit of given number is 0 or 1.
    - Example: Check bit 3 of a number.

```
num     (9) 1001        (6) 0110
& 8         1000            1000
-----------------------------------------------------
            1000            0000
        non-zero - (1)      zero - (0)
```

    - Example: Check bit 2 of a number.

```
num     (9) 1001        (6) 0110
& 4         0100            0100
-----------------------------------------------------
            0000            0100
            zero - (0)      non-zero - (1)
```

```c
int main() {
    int num, n;
    printf("enter a number: ");
    scanf("%d", &num);
    printf("check which bit (LSB=0): );
    scanf("%d", &n);
    if( num & (1 << n) )
        printf("%dth bit is 1\n", n);
    else
        printf("%dth bit is 0\n", n);
    return 0;
}
```

- Print the binary value for a given int.

```c
void print_bin(int num) {
    int n;
    for(n = 31; n >= 0; n--) {
        if( num & (1 << n) )
            putchar('1');
        else
            putchar('0');
    }
    printf("\n");
}
int main() {
    int num;
    printf("enter a number: ");
    scanf("%d", &num);
    print_bin(num);
    return 0;
}
```

- Print the binary representation any value (of any type).

```c
void print_bin(void *ptr, int size) {
    unsigned char *pchar = (unsigned char *)ptr;
    unsigned char num;
    int i, n;
    for(i=0; i<size; i++) {
        num = *(pchar + i);
        for(n = 7; n >= 0; n--) {
            if( num & (1 << n) )
                putchar('1');
            else
                putchar('0');
        }
```

```
            putchar(' ');
        }
        printf("\n");
    }
    int main() {
        int a = 65;
        float b = 123.45;
        long c = 9;
        //struct emp e = { ... };
        print_bin(&a, sizeof(a));
        print_bin(&b, sizeof(b));
        print_bin(&c, sizeof(c));
        //print_bin(&e, sizeof(e));
        return 0;
    }
```

- Most frequently used bitwise operations in micro-controller programming.
    - Helper macro

```
#define BV(n) (1 << (n))
```

    - Check nth bit of a register.

```
if(regr & BV(n))
    printf("nth bit = 1.\n");
else
    printf("nth bit = 0.\n");
```

    - Clear nth bit of a register.

```
num:    xxxx xxxx
& ~8    1111 0111
-------------------
        xxxx 0xxx
```

```
regr = regr & ~BV(n);
// OR
regr &= ~BV(n);
```

    - Set nth bit of a register.

```
num:    xxxx xxxx
| 8     0000 1000
------------------
        xxxx 1xxx
```

```
regr = regr | BV(n);
// OR
regr |= BV(n);
```

- Toggle nth bit of a register.

```
num:    xxxx xxxx
^ 8     0000 1000
------------------
        xxxx x'xxx
```

```
regr = regr ^ BV(n);
// OR
regr ^= BV(n);
```

- Assignment (Try the following without changing other bits):
  - Set bit 20, 23, and 29 of given register.
  - Clear bit 5 and 12 of given register.
  - Toggle bit 9 of given register.
  - Wait while bit 31 of given register is 0.
  - Wait while bit 6 of given register is 1.
  - Set value of "num" in last 4 bits of given register. Assume that num < 16.
  - Set value of "num" in given register bit 12-15. Assume that num < 16.

Structures

Tagged initialization

- Feature of C99 standard

```
struct emp {
    int id;
    char name[20];
    double sal;
    char dept[20];
    struct date {
        int day, month, year;
```

```
    } joining;
};

int main() {
    struct emp e1 = { 1, "Nilesh", 2000.00, "Training", { 31, 05, 2004 } };
    struct emp e2 = { 1, "Nilesh" }; // init id and name. Rest elements are
0.
    struct emp e3 = { 0, "", 2000.00, "Training" }; // init sal and dept.
Rest elements are 0.

    // taged initialization
    struct emp e4 = {
        .id = 1,
        .name = "Nilesh",
        .sal = 2000.00,
        .dept = "Training",
        .joining = { .day = 31, .month = 05, .year = 2004 }
    };

    struct emp e5 = {
        .id = 1,
        .dept = "Training",
        .joining = { .day = 31, .month = 05, .year = 2004 },
        .sal = 2000.00,
        .name = "Nilesh"
    };

    struct emp e6 = {
        .sal = 2000.00,
        .dept = "Training"
    }; // init sal and dept. Rest elements are 0.

    return 0;
}
```

Dot and Arrow operator

- `arr[i] = *(arr + i)`
- `var.member = ???`
- `ptr->member = ???`