Лекция 19. Побитовые операции (bitwise operation) Часть 3.

Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

Курс «Программирование», весенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)

Содержание

- Операции: ~, &, |, ^, >> , <<, &=, |=, ^=, >>=, <<=
- Обзор двоичной, восьмеричной и шестнадцатеричной систем счисления
- Два средства языка С для обработки отдельных битов значения: побитовые операции и битовые поля
- Ключевые слова: _Alignas, _Alignof
- Примеры “структуры” на целочисленном типе данных с использованием побитовых операций.

Побитовые операции (1)

Операция дополнение до единицы или **побитовое отрицание: ~** (тильда)

Унарный оператор ~ преобразует каждую единицу в ноль, а каждый ноль в единицу.

```
                    int val=100; //0110 0100
~(1010 1010)        val= ~(val); //~(0110 0100)
 ( 101 0101)        printf(val)  //-101 (1001 1011)
```

Вспоминаем про преобразование знаковых и беззнаковых чисел.

Побитовые операции (2)

Побитовая операция "И": & (амперсант)

```
int num1 = 100; //01100100
int num2 = 192; //11000000
printf("%d",num1&num2); // 64
```

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

Побитовые операции (3)

Побитовая операция "ИЛИ": | (терминатор)

```
int num1 = 100; //01100100
int num2 = 153; //10011001
printf("%d",num1|num2); // 253
```

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

Побитовые операции (4)

Побитовая операция исключающее "ИЛИ": ^ (циркумфлекс)

```
int num1 = 116; //01110100
int num2 = 249; //11111001
printf("%d",num1^num2); // 141
```

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

Побитовые операции (5)

Побитовая бинарная операция сдвиг влево "<<"

До сдвига	0	1	0	0	0	1	1	1
Результат	0	0	0	1	1	1	0	0

```
flags = flags << 2
```

71, 0107, 0x47

28, 034, 0x1C

Побитовые операции (6)

Побитовая бинарная операция сдвиг влево ">>"

До сдвига	0	1	0	0	0	1	1	1
Результат	0	0	0	1	0	0	0	1

```
flags = flags >> 2
```

71, 0107, 0x47

17, 021, 0x11

Побитовые операции (7)

Для всех побитовых операций доступны сокращенные формы записи скомбинированные с оператором присваивания.

```
val &= 0377;
val |= 0377;
val ^= 0377;
color >>= 8;
color <<= 8;
```

Средства выравнивания - C11 (1)

Средства **выравнивания** по своей природе больше ориентированы на манипулирование ***байтами***, чем ***битами***, но они также отражают возможность языка Си при работе с аппаратной составляющей. В этом контексте ***выравнивание*** относится к тому, как объекты располагаются в памяти.

Пример, для максимальной эффективности система может требовать, чтобы значение типа **double** хранилось в памяти по адресу, кратному 4, но разрешать значению типа **char** храниться по любому адресу.

Средства выравнивания - C11 (2)

Операция `_Alignof` выдает требования к выравниванию указанного типа. Для ее использования необходимо после ключевого слова `_Alignof` поместить имя типа в круглых скобках.

```
size_t f_align = _Alignof(float);
```

Адрес: 0x1000 0x1001 0x1002 0x1003 0x1004 0x1005 0x1006 0x1007 0x1008…
Ячейки: [float] [char] [padding] [float] ...

Средства выравнивания - C11 (3)

```
struct Foo {
    char a;      // 1 байт
    float b;     // 4 байта
    short c;     // 2 байта
};
_Alignof(char); //Выравнивание char: 1 байт
_Alignof(float); //Выравнивание float: 4 байт
_Alignof(short); //Выравнивание short: 2 байт
```

Средства выравнивания - C11 (4)

В качестве значения для выравнивания берётся степень двойки, более **высокие** значения имеют более *строгий* или более *жесткий*, в отличии от более **низкие** значений, в то время как более **низкие** значения трактуются, как более *слабые*.

Более строгое вырывание для типа данных или для переменной можно получить при помощи спецификатора `_Alignas`. При помощи `_Alignas` можно запрашивать любое выравнивание.

Средства выравнивания - C11 (5)

При помощи `_Alignas` можно запрашивать любое выравнивание. Однако не стоит запрашивать выравнивание которое слабее фундаментального выравнивания, принятого для типа данных.

Например: фундаментально `float` выравнивается по 4 байта, не стоит запрашивать выравнивание, равное 1 или 2 байта.

```
_Alignas(double) char c1;
_Alignas(8) char c2;
```

Средства выравнивания - C11 (6)

```
double dx;
char ca, cx;
double dz;
char cb;
char _Alignas(double) cz;
printf("Выравнивание char:%zd\n",
_Alignof(char));//Выравнивание char:1
printf ("Выравнивание double:%zd\n",
_Alignof(double));//Выравнивание double:8
```

Средства выравнивания - C11 (7)

```
printf("&dx: %p\n", &dx);0x7fff5fbff660
printf("&ca: %p\n", &ea);0x7fff5fbff65f
printf("&cx: %p\n", &ex);0x7fff5fbff65e
printf("&dz: %p\n", &dz);0x7fff5fbff650
printf("&cb: %p\n", &eb);0x7fff5fbff64f
printf("&cz: %p\n", &cz);0x7fff5fbff648
```

Средства выравнивания - C11 (8)

0000007E9FDFF710

&dx: 710, &ca: 720, &cx: 721,

&dz: 718, &cb: 722, &cz: 708

Адрес	708	709	70A	70B	70C	70D	70E	70F
Ячейки	cz	Отступы выравнивания						
Адрес	710	711	712	713	714	715	716	717
Ячейки	dx							
Адрес	718	719	71A	71B	71C	71D	71E	71F
Ячейки	dz							
Адрес	720	721	722	723	724	725	726	727
Ячейки	ca	cx	cb					

Средства выравнивания - C11 (9)

0000007E9FDFF710

&dx: 710, &ca: 70D, &cx: 70E,

&dz: 718, &cb: 70F, &cz: 708

Адрес	708	709	70A	70B	70C	70D	70E	70F
Ячейки	cz					ca	cx	cb
Адрес	710	711	712	713	714	715	716	717
Ячейки	dx							
Адрес	718	719	71A	71B	71C	71D	71E	71F
Ячейки	dz							

Средства выравнивания - C11 (10)

0000007E9FDFF710

&dx: 710, &ca: 70D, &cx: 70E,

&dz: 718, &cb: 70F, &cz: 708

Адрес	708	709	70A	70B	70C	70D	70E	70F
Ячейки	cz					ca	cx	cb
Адрес	710	711	712	713	714	715	716	717
Ячейки	dx							
Адрес	718	719	71A	71B	71C	71D	71E	71F
Ячейки	dz							

Применение выравнивания (1)

Применение выравнивания чаще всего связано с использованием строгости в при передачи данных в различных **сетевых протоколах**, а также в технологиях связанных с аппаратной реализацией **CPU** и **GPU**.

Применение выравнивания (2)

Выравнивание структур

```
Размер структуры: 24
Выравнивание структуры: 8
```

```c
typedef struct {
    char a;
    _Alignas(8) int b;
    double c;
} FooStuct;
FooStuct s;
printf("Размер структуры: %zu\n", sizeof(s));
printf("Выравнивание структуры: %zu\n",
_Alignof(FooStuct));
```

Применение выравнивания (3)

Векторизация

```
#include <immintrin.h> // Для векторных инструкций

_Alignas(32) float vec[8]; // Выровненный массив для AVX
__m256d vec_reg = _mm256_load_pd(vec);
```

Применение выравнивания (4)

Выравнивание данных под кэш-линию

```
typedef struct _Alignas(144)
                        CacheAlignedStruct {
    int data[36];
} CacheAlignedStruct;
```

Кэш L1:	1,1 МБ
Кэш L2:	9,0 МБ
Кэш L3:	18,0 МБ

Применение выравнивания (5)

Работа с GPU (CUDA/OpenCL)

```
#include <cuda_runtime.h>
#include <stdalign.h>

_Alignas(128) float cudaArray[1024]
 cudaMemcpyToSymbol(cudaArray, ...);
```

Применение выравнивания (6)

В драйверах устройств (аппаратные требования)

```
typedef struct {
_Alignas(4) volatile uint32 tcontrol_register;
_Alignas(4) volatile uint32 t status_register;
} HardwareRegisters;
HardwareRegisters *regs =
(HardwareRegisters*)0x1000;
regs->control_register = 0x12345678;
```

Применение выравнивания (7)

Union с разными типами

```
typedef union {
    _Alignas(double) double d;
    _Alignas(int) int i;
    _Alignas(char) char c[8]; // Размер совпадает с double
} MixedUnion;
```

Пример: побитовая структура (1)

```
typedef struct bitlaptop{
    unsigned int cpuType: 2;
    unsigned int ramV: 3;
    unsigned int hasGPU: 1;
    unsigned int osSet: 2;
    unsigned int storT: 2;
    unsigned int storV: 4;
    unsigned int screenS: 2;
} BITLAPTOP;
```

Пример: побитовая структура (2)

```
typedef unsigned int pbLaptop;
#define CPU_T_MASK     0x0003// 2 бита
#define RAM_V_MASK       0x001C// 3 бита
#define HAS_GPU_MASK     0x0020// 1 бит
#define OS_SET_MASK      0x00C0// 2 бита
#define STOR_T_MASK      0x0003// 2 бита
#define STOR_V_MASK      0x003C// 4 бита
#define SCREEN_S_MASK    0x0FC0// 2 бит
```

Пример: побитовая структура (3)

```
1111 1111 1111 1111 1111 1111 1111 1111
CPU_T_MASK      0000 0000 0000 0011
RAM_V_MASK      0000 0000 0001 1100
HAS_GPU_MASK    0000 0000 0010 0000
OS_SET_MASK     0000 0000 1100 0000
STOR_T_MASK     0000 0011 0000 0000
STOR_V_MASK     0011 1100 0000 0000
SCREEN_S_MASK   1100 0000 0000 0000
```

Пример: побитовая структура (4)

```
#define RAM_V_SHIFT          2
#define HAS_GPU_SHIFT        5
#define OS_SET_SHIFT         6
#define STOR_T_SHIFT         8
#define STOR_V_SHIFT         10
#define SCREEN_S_SHIFT       14
```

Пример: побитовая структура (5)

```
static char *cpu[] = {"Intel", "AMD", "Apple
Silicon"};
static const int ramVol[] = {8, 16, 32, 64, 128};
static const char *os[] = {"Linux", "macOS",
"Windows"};
static const char *storageT[] = {"HDD", "SSD",
"SSD+NVMe"};
static const int storageV[] = {128, 256, 512,
1024, 2048};
static const char *screen[] = {"<14''",">14'' &&
<=15''",">15'' && <=16''",">=17''"};
```

Пример: побитовая структура (6)

```
typedef struct bitlaptop{
    unsigned int cpuType: 2;        //0 - 00
    unsigned int ramV: 3;           //1 - 001
    unsigned int hasGPU: 1;         //0 - 0
    unsigned int osSet: 2;          //2 - 10
    unsigned int storT: 2;          //1 - 01
    unsigned int storV: 4;          //2 - 0010
    unsigned int screenS: 2;         //2 - 10
} BITLAPTOP;
   0000 0000 0000 0000 1000 1001 1000 0100
```

Пример: побитовая структура (7)

Извлечение данных

```
1000 1001 1000 0100
int cpuType = (specs1 & CPU_T_MASK);
(1000 1001 1000 0100 & 11) = 0000 0000 0000 0000
int storageSize = (specs1 & STOR_V_MASK) >>
                                    STOR_V_SHIFT;
1000 1001 1000 0100 & 0011 1100 0000 0000 =
= 0000 1000 0000 0000 >> 10 =
                    = 0000 0000 0000 0010
```

Пример: побитовая структура (8)

```
int cpuType = (specs1 & CPU_T_MASK) >> CPU_T_SHIFT;
int ramSize = (specs1 & RAM_V_MASK) >> RAM_V_SHIFT;
bool hasGPU = (specs1 & HAS_GPU_MASK) >>
                                        HAS_GPU_SHIFT;
int osFamily = (specs1 & OS_SET_MASK) >>
                                        OS_SET_SHIFT;
int storageType = (specs1 & STOR_T_MASK) >>
                                        STOR_T_SHIFT;
int storageSize = (specs1 & STOR_V_MASK) >>
                                        STOR_V_SHIFT;
int screenSizeInch = (specs1 & SCREEN_S_MASK) >>
                                        SCREEN_SSHIFT;
```

Пример: побитовая структура (7)

Запись данных

```
pbLaptop set_cpu_t(pbLaptop current, int value){
    if (value < 0 || value > 2) return current;
    return (current & ~CPU_T_MASK) | ((value << 0)
& CPU_T_MASK);
}
```

value = 3; (11)

1000 1001 1000 0100 & ~(0000 0000 0000 0011)
1000 1001 1000 0100 & 1111 1111 1111 1100 =

= 1000 1001 1000 0100 | (11 << 0) =

=1000 1001 1000 0111

Пример: побитовая структура (8)

```
pbLaptop set_ram_v(pbLaptop current, int value){
    if (value < 0 || value > 3) return current;
    return (current & ~RAM_V_MASK) | ((value << RAM_V_SHIFT) & RAM_V_MASK);
}
value = 4; (100)
1000 1001 1000 0111 & ~(0000 0000 0001 1100)
1000 1001 1000 0111 & 1111 1111 1110 0011 =
= 1000 1001 1000 0011 | (100 << 2) =1 0000
= 1000 1001 1001 0011
```

Спасибо за внимание

Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

Курс «Программирование», весенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)