ПРОГРАММИРОВАНИЕ

Битовые операции

Преподаватель:

Ст. преп. Кафедры ВС,

Перышкова Евгения Николаевна

Язык C и битовые операции

Язык C поддерживает все существующие битовые операции.

Создавался чтобы заменить ассемблер.

Если не применять оптимизацию, можно оценить, в какие конструкции преобразуется код программы

Битовые операции

Битовые операции — это тестирование, установка или сдвиг битов в байте или слове, которые соответствуют стандартным типам языка C char и int.

Битовые операторы не могут использоваться с float, double, long double и другими сложными типами.

Логические операции

Логические операции И, ИЛИ, исключающее ИЛИ и НЕ могут быть описаны с помощью таблиц истинности.

Логический оператор И		
X	Y	X AND Y
0	0	0
1	0	0
0	1	0
1	1	1

Логический оператор ИЛИ		
X	Y	X OR Y
0	0	0
1	0	1
0	1	1
1	1	1

Логические операции

Логические операции И, ИЛИ, исключающее ИЛИ и НЕ могут быть описаны с помощью таблиц истинности.

Логический оператор исключающее ИЛИ		
X	Y	X XOR Y
0	0	0
1	0	1
0	1	1
1	1	0

Логический оператор НЕ	
X	NOT X
0	1
1	0

Битовые операторы

Битовые операторы И, ИЛИ, НЕ используют ту же таблицу истинности, что и их логические эквиваленты, за тем исключением, что они работают побитно.

В побитовых операциях значение бита, равное 1, рассматривается как логическая истина, а 0 как ложь. Побитовое И (оператор &) берёт два числа и логически умножает соответствующие биты.

Таблица битовых операторов

Оператор	Действие
&	И
|	ИЛИ
^	Исключающее ИЛИ
~	Дополнение
>>	Сдвиг вправо
<<	Сдвиг влево

Битовые операторы

Битовые операторы наиболее часто применяются при разработке драйверов устройств, например программ для модемов, дисков и принтеров, поскольку битовые операторы могут использоваться для выключения некоторых битов, например четности.

Бит четности используется для подтверждения того, что остальные биты в байте не изменялись. Он, как правило, является старшим битом в байте.

Битовые операторы

Битовое И чаще всего используется для выключения битов

Любой бит, установленный в 0, вызывает установку соответствующего бита в другом операнде также в 0.

Битовые операторы

Функция читает символы из порта модема, используя функцию read_modem(), и сбрасывает бит четности в 0.

```
char get_char_from_modem()
{
char ch;
ch = read_modem (); /* получение символа из порта модема * /
return (ch & 127);
}
```

Битовые операторы

бит четности	
|	
11000001	ch содержит 'A' с битом четности
01111111	127 в двоичном представлении выполнение битового И
&----------	
01000001	'A' без бита четности

Битовые операторы

Битовое ИЛИ может использоваться для установки битов.

Любой бит, установленный в любом операнде, вызывает установку соответствующего бита в другом операнде.

Результат операции 128 | 3	
10000000	128 в двоичном представлении
00000011	3 в двоичном представлении
|----------	битовое ИЛИ
10000011	результат

Битовые операторы

Исключающее ИЛИ (XOR) устанавливает бит, если соответствующие биты в операндах отличаются.

Результат операции 127 ^ 120	
01111111	127 в двоичном представлении
01111000	120 в двоичном представлении
^----------	исключающее ИЛИ
00000111	результат

Битовые операторы

Операторы сдвига >> и << сдвигают биты в переменной вправо и влево на указанное число.

Общий вид оператора сдвига вправо:
переменная >> число сдвигов

Общий вид оператора сдвига влево:
переменная << число сдвигов

Битовые операторы

Операции битового сдвига могут быть полезны при декодировании информации от внешних устройств и для чтения информации о статусе.

Операторы битового сдвига могут также использоваться для выполнения быстрого умножения и деления целых чисел. Сдвиг влево равносилен умножению на 2, а сдвиг вправо - делению на 2

Битовые операторы

	Битовое представление x после выполнения каждого оператора	**Значение x**
char x;		
x = 7;	00000111	7
x = x << 1;	00001110	14
x = x << 3;	01110000	112
x = x << 2;	11000000	192
x = x >> 1;	01100000	96
x = x >> 2;	00011000	24

Битовые операторы

Оператор дополнение (~) инвертирует состояние каждого бита указанной переменной, то есть 1 устанавливается в 0, а 0 — в 1.

Битовые операторы часто используются в процедурах шифрования.

Результат операции ~	
00101100	Исходный байт
11010011	После первого дополнения
00101100	После второго дополнения

Битовые операторы

```
/* Простейшая шифрующая функция */

char encode(code ch)
{
return (~ch); /* дополнение */
}
```

Битовые и логические операторы

```
int a = 3;
int b = 4;
printf("a & b = %d\n", a & b);
printf("a && b = %d\n", a && b);
```

Применение битовых операций

Двоичное представление IP адреса узла и маски

192.168.234.35
1100 0000.1010 1000.1110 1010.0010 0011

255.255.255.224
1111 1111.1111 1111.1111 1111.1110 0000

Применение битовых операций

Получение адреса сети:

1100 0000.1010 1000.1110 1010.0010 0011
&
1111 1111.1111 1111.1111 1111.1110 0000
1100 0000.1010 1000.1110 1010.0010 0000

1100 0000.1010 1000.1110 1010.0010 0000
192.168.234.32

Применение битовых операций

Инвертируем маску:

1111 1111.1111 1111.1111 1111.1110 0000
~
0000 0000.0000 0000.0000 0000.0001 1111

Применение битовых операций

Получение адреса узла:

1100 0000.1010 1000.1110 1010.0010 0011
&
0000 0000.0000 0000.0000 0000.0001 1111
0000 0000.0000 0000.0000 0000.0000 0011

0000 0000.0000 0000.0000 0000.0000 0011
0.0.0.3