Лекция 5.
Язык программирования - ассемблер.
Логические основы компьютера

Булева алгебра

Виды логических операций

Для логических величин обычно используются три операции:

- Конъюнкция – логическое умножение (И) – and, &, ∧
- Дизъюнкция – логическое сложение (ИЛИ) – or, |, V.
- Унарная операция – логическое отрицание (НЕ) – not, ¬.

Основные аксиомы

Булевой алгеброй называется непустое множество A с двумя бинарными операциями - конъюнкцией и дизъюнкцией, а также унарной операцией и двумя выделенными элементами: 0 (или Ложь) и 1 (или Истина) такими, что для всех a, b и c из множества A верны следующие аксиомы:

$a \vee (b \vee c) = (a \vee b) \vee c$	$a \wedge (b \wedge c) = (a \wedge b) \wedge c$	ассоциативность
$a \vee b = b \vee a$	$a \wedge b = b \wedge a$	коммутативность
$a \vee (a \wedge b) = a$	$a \wedge (a \vee b) = a$	законы поглощения
$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$	$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$	дистрибутивность
$a \vee \neg a = 1$	$a \wedge \neg a = 0$	дополнительность

Основные свойства

Для всех a и b из множества A верны следующие равенства:

$a \vee a = a$	$a \wedge a = a$	
$a \vee 0 = a$	$a \wedge 1 = a$	
$a \vee 1 = 1$	$a \wedge 0 = 0$	
$\neg 0 = 1$	$\neg 1 = 0$	дополнение 0 есть 1 и наоборот
$\neg(a \vee b) = \neg a \wedge \neg b$	$\neg(a \wedge b) = \neg a \vee \neg b$	законы де Моргана
$\neg\neg a = a.$		инволютивность отрицания

Основные свойства

Самая простая нетривиальная булева алгебра содержит всего два элемента, 0 и 1, а действия в ней определяются следующей таблицей:

Конъюнкция

∧	0	1
0	0	0
1	0	1

Дизъюнкция

∨	0	1
0	0	1
1	1	1

Унарная операция

a	0	1
¬a	1	0

Таблицы истинности:

a	b	$a \wedge b$
0	0	0
0	1	0
1	0	0
1	1	1

a	b	$a \vee b$
0	0	0
0	1	1
1	0	1
1	1	1

Логические элементы

Вентиль - это устройство, которое выдает результат булевой операции от введенных в него данных (сигналов).

Простейший вентиль представляет собой транзисторный инвертор, который преобразует низкое напряжение в высокое или наоборот. Это можно представить как преобразование логического нуля в логическую единицу или наоборот. Т.е. получаем вентиль НЕ.

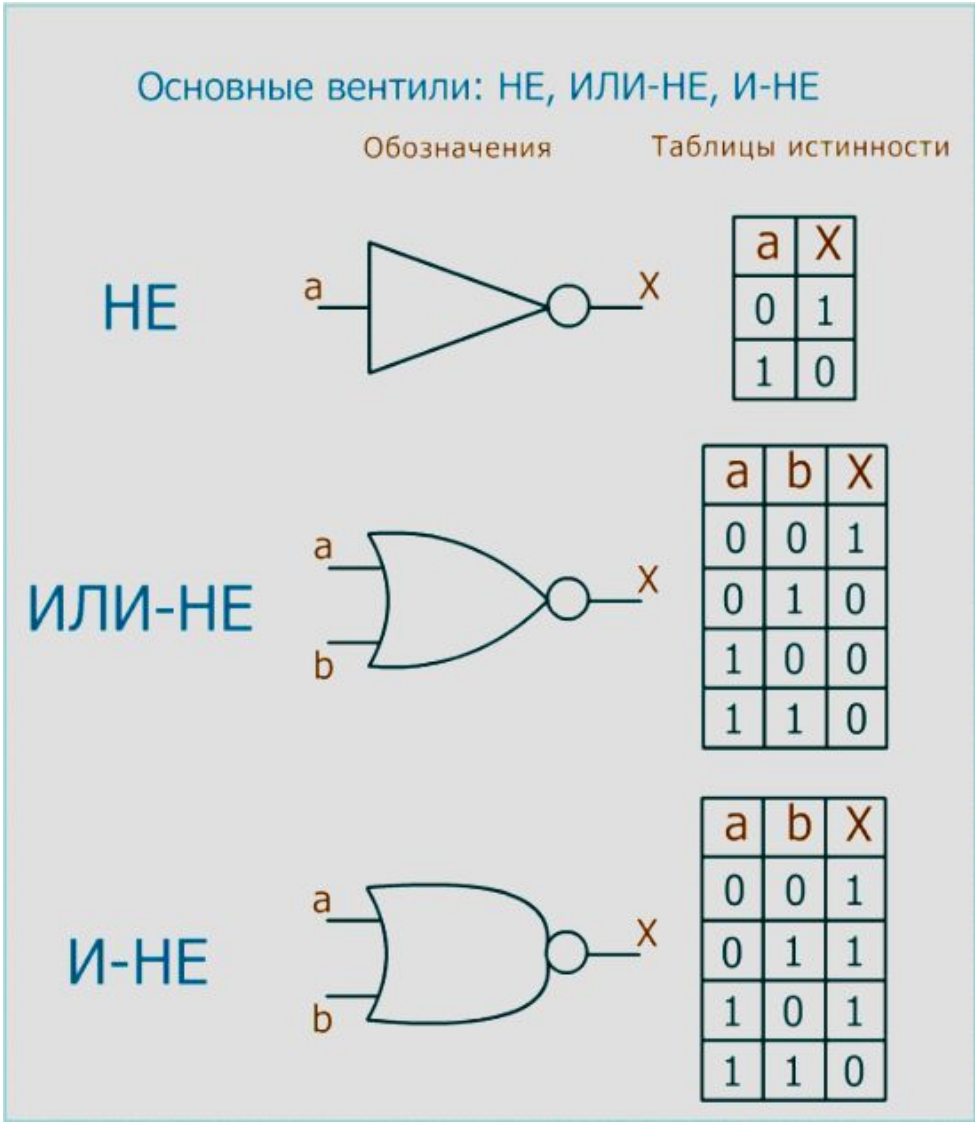
Соединив пару транзисторов различным способом, получают вентили ИЛИ-НЕ и И-НЕ. Эти вентили принимают уже не один, а два и более входных сигнала. Выходной сигнал всегда один и зависит (выдает высокое или низкое напряжение) от входных сигналов.

В случае вентиля ИЛИ-НЕ получить высокое напряжение (логическую единицу) можно только при условии низкого напряжении на всех входах.

В случае вентиля И-НЕ логическая единица получается, если все входные сигналы или хотя бы один из них будут нулевыми.

Транзистору требуется очень мало времени для переключения из одного состояния в другое (время переключения оценивается в наносекундах). И в этом одно из существенных преимуществ схем, построенных на их основе.

Логические элементы



Другие логические элементы:

- Сумматор
- Полусумматор
- Триггер

Триггеры и сумматоры – это относительно сложные устройства, состоящие из более простых элементов – вентилей.

Триггер способен хранить один двоичный разряд, за счет того, что может находиться в двух устойчивых состояниях. В основном триггеры используется в регистрах процессора.

Сумматоры широко используются в арифметико-логических устройствах (АЛУ) процессора и выполняют суммирование двоичных разрядов.

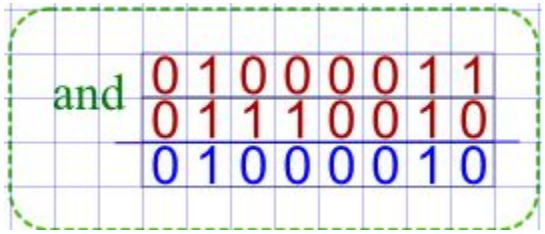
Битовые операции

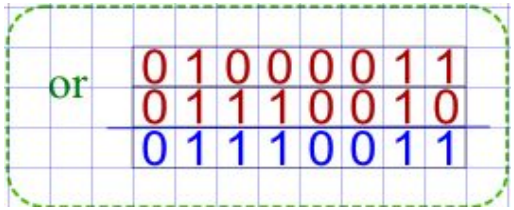
1. Переведем пару произвольных целых чисел до 256 (один байт) в двоичное представление.

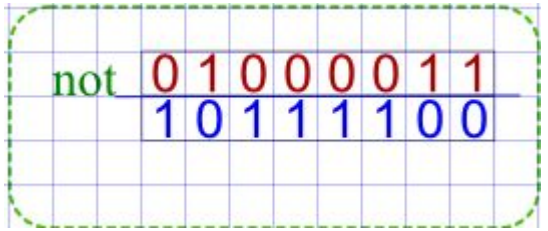
$67_{10} = 0100\ 0011_2$

$114_{10} = 0111\ 0010_2$

2. Расположим биты второго числа под соответствующими битами первого и выполним обычные логические операции к цифрам, стоящим в одинаковых разрядах первого и второго числа.







3. Переведем результат в десятичную систему счисления.

$01000010 = 2^6 + 2^1 = 64 + 2 = 66$

$01110011 = 2^6 + 2^5 + 2^4 + 2^1 + 2^0 = 64 + 32 + 16 + 2 + 1 = 115$

$10111100 = 2^7 + 2^5 + 2^4 + 2^3 + 2^2 = 128 + 32 + 16 + 8 + 4 = 188$

4. В результате побитовых логических операций получилось следующее:

67 and 114 = 66

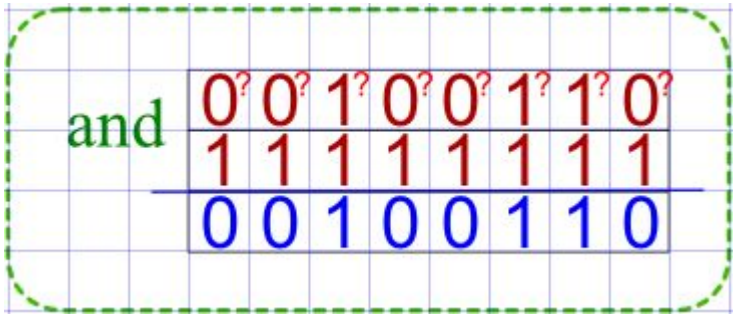
67 or 114 = 115

not 67 = 188

Назначение побитовых логических операций

Проверка битов

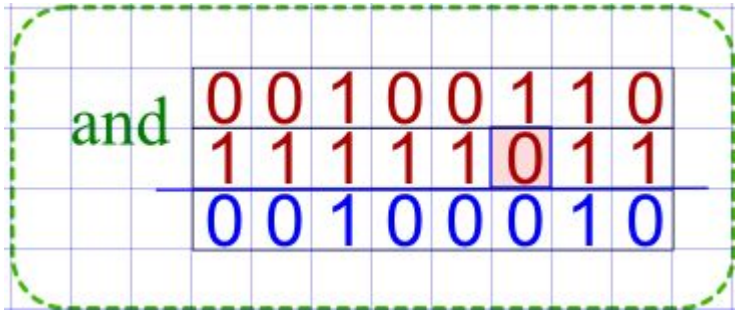
Другими словами, x and 255 = x.



Обнуление битов

Чтобы обнулить какой-либо бит числа, нужно его логически умножить на 0.

```
1111 1110 = 254 = 255 - 1 = 255 - 2^0
1111 1101 = 253 = 255 - 2 = 255 - 2^1
```

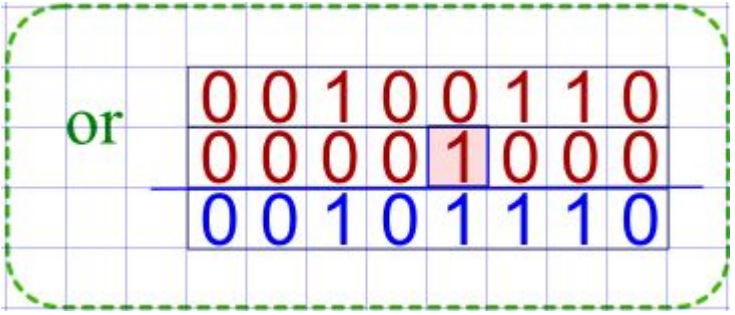


Т.е. чтобы обнулить четвертый с конца бит числа x, надо его логически умножить на 247 или на (255 - 2^3).

Установка битов в единицу

Для установки битов в единицу используется побитовая логическая операция ИЛИ.

```
0000 0001 = 2^0 = 1
0000 0010 = 2^1 = 2
```



Чтобы установить второй по старшинству бит числа x в единицу, надо его логически сложить с 64 (x or 64).

Язык программирования АССЕМБЛЕР

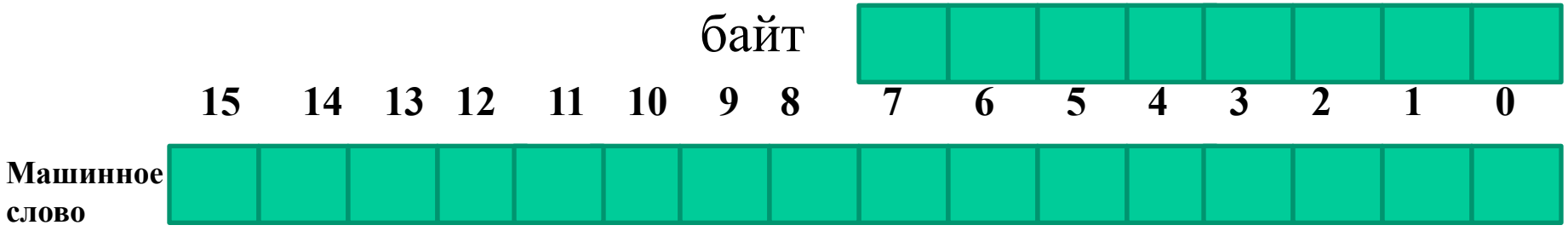
Assembler — язык программирования низкого уровня, представляющий собой формат записи машинных команд, удобный для восприятия человеком.

Команды языка ассемблера один в один соответствуют командам процессора и, фактически, представляют собой удобную символьную форму записи (мнемокод) команд и их аргументов. Также язык ассемблера обеспечивает базовые программные абстракции: связывание частей программы и данных через метки с символьными именами и директивы.

Представление данных в компьютере

Директивы ассемблера позволяют включать в программу блоки данных (описанные явно или считанные из файла); повторить определённый фрагмент указанное число раз; компилировать фрагмент по условию; задавать адрес исполнения фрагмента, менять значения меток в процессе компиляции; использовать макроопределения с параметрами и др.

Каждая модель процессора, в принципе, имеет свой набор команд и соответствующий ему язык (или диалект) ассемблера.



Двойное машинное слово - 32 бита (4 байта)

Системы счисления

Десятичная	Двоичная	Шестнадца-теричная
0	0000	00
1	0001	01
2	0010	02
3	0011	03
4	0100	04
5	0101	05
6	0110	06
7	0111	07
8	1000	08
9	1001	09
10	1010	0A
11	1011	0B
12	1100	0C
13	1101	0D
14	1110	0E
15	1111	0F
16	10000	10

В позиционных системах счисления, к которым относятся и широко распространенная десятичная система, числовое значение цифры зависит от ее местоположения или позиции в последовательности цифр изображающих число.

Единственной, дошедшей до нашего времени, системой, не относящейся к позиционной системе счисления, является римская система счисления.

Любое число в позиционной системе счисления изображается последовательностью цифр:

$$X = a_{n-1}\, a_{n-2} \ldots a_1 a_0 ,$$

где $a_i \in \{0,1,\ldots,q-1\}$, q – основание системы счисления.

Перевод чисел из одной системы счисления в другую

Правило №1: Десятичное значение числа, записанного в любой системе исчисления, определяется по формуле:

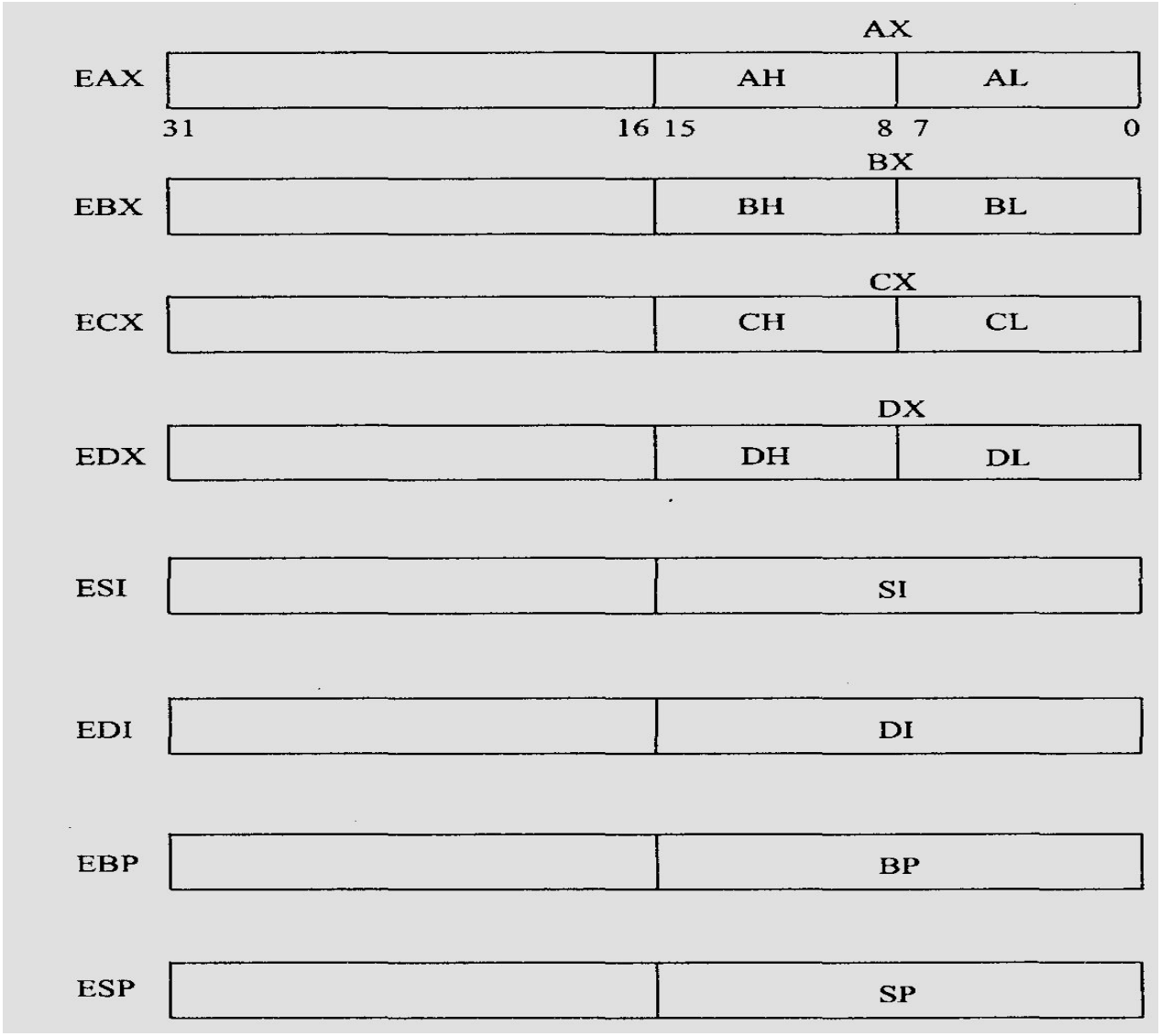
$$X_{10} = a_{n-1}..a_{0q} = \sum_{i=0}^{n-1} a_i \cdot q^i,$$

где a_i - разряды исходного числа; q^i - вес i-ого разряда; i – номер текущего разряда; n – число разрядов исходного числа.

Правило №2: Для перевода целого десятичного числа X в систему счисления с основанием q необходимо последовательно делить исходное число X и образующиеся частные на основание q до получения частного равного нулю. Искомое представление числа есть последовательность остатков от операций деления, причем первый остаток дает младшую цифру искомого числа.

Правило №3: Для перевода двоичного числа в систему счисления с основанием 16 необходимо исходное число справа налево сгруппировать по 4 бита, а затем каждую группу записать одной шестнадцатеричной цифрой.

Регистры общего назначения



Аккумулятор

Базовый регистр

Регистр-счетчик

Регистр данных

Индекс источника

Индекс приемника

Указатель стека

Указатель базы

Синтаксис языка ассемблер

Общепринятого стандарта для синтаксиса языков ассемблера не существует. Однако, существуют стандарты де-факто — традиционные подходы, которых придерживаются большинство разработчиков языков ассемблера. Основными такими стандартами являются Intel-синтаксис и AT&T-синтаксис.

Общий формат записи инструкций одинаков для обоих стандартов: `[метка:] опкод [операнды] [;комментарий]`

Опкод — непосредственно мнемоника инструкции процессору. К ней могут быть добавлены префиксы (повторения, изменения типа адресации и пр.). В качестве операндов могут выступать константы, названия регистров, адреса в оперативной памяти и пр.. Различия между стандартами Intel и AT&T касаются, в основном, порядка перечисления операндов и их синтаксиса при различных методах адресации.