1. **Как представляется символьная информация в компьютере в кодах ASCII, расширениях ASCII и различных кодировках Unicode?**
2. **Как хранятся русские буквы в «классических» и широких строках**

???
* широкая строка - представленная символами юникода ?
* классическая - ascii?
???

ASCII - старейшая кодовая таблица описывает цифры, строчные и прописные буквы латиницы, некоторые знаки препинания и специальные символы(ASCII символы ) и сопоставляет им коды в диапазоне от 0 до 127. Все современные таблицы основаны н ASCII

Сейчас обычно используется Unicode, сопоставляющий кириллице коды от 1024 до 1279 (обычно юникод записывают в шестнадцатеричном виде - от 0400 до 04FF )

Для того, чтобы представление буквы кириллицы в памяти компьютера не могло совпасть с представлением последовательности из нескольких ASCII-символов, используются различные кодировки Unicode.

Наиболее распространённая из них — UTF-8 — записывает символ в виде цепочки байтов, включающих, кроме собственно кода, ещё и служебную информацию. Соответственно, кириллица, представленная в UTF-8, занимает диапазон от D080 до D19F, так что кириллические буквы занимают два байта.

**3.Как представляются целые числа со знаком и без знака?**

Без знака - бинарный код, с учетом того, что число минимально возможного размера называется байтом. Ячейка памяти не может быть пуста, в ней содержится либо 0, либо 1.
Минимальное число = 0
Максимальное 2^N - 1
109 = 1101101 (двоичное) = 01101101 в байте (8-битном)

Со знаком:
Есть несколько способов представить отрицательное число:

1. Величина со знаком
2. Код с избытком 128
3. Дополнительный код (дополнение до 2)

Величина со знаком:
Выделяется старший бит под знак
(мы не можем представить число -128 или 128  в 8 битах)
-127 = 1111 1111

Код с избытком

u = x+KSI // соответствующее беззнаковое значение
x = u -KSI

KSI =128

-128 = 0000 0000
0 = 1000 0000
127 = 1111 1111 // как 255

можно использовать беззнаковый сумматор, но нужна коррекция
res = (x+KSI) + (y+KSI) = (x+y) + 2KSI
// нужно вычитание
res -= KSI
т.е. нужно сложение и вычитание

Дополнительный код:
(Дополнение до 2)

Возможность пользоваться беззнаковым сумматором без коррекции результата
-x = 0 - x = 2^N - x
Этот код называется дополнительным
На практике делают так:
-x = 0-x = (-1-x)+1
-1-x // инверсия (-1 в дом коде  == 1111 1111)
+1 // получим дополнительный код

нет способа понять -1 или 255 записано в ячейку

## 5. Как выполняются логические операции  и сдвиги над строкой битов? Расширения, зачем нужны и т.д.

Логические операции :
- нет переноса между разрядами
- не забываем про ведущие единицы (при логическом отрицании )
- конъюнкция: 3&5 = 0000 0011 & 0000 0101 = 0000 0001 = 1
- аналогично дизъюнкция
- и xor - 3^5 = 0000 0011^0000 0101 = 1111 0110 = -10 = 246

Расширение - увеличение разрядности числа

```
int i;
 short int s = -1;
 i = s;
```

Младшие n  разрядов совпадают с расширенным значением X, старшие m-n должны быть как-то  инициализированы

-Беззнаковое расширение : расширяемая часть заполняется 0 (сохраняет значение беззнаковой интерпретации x)

- Знаковое расширение  - расширяемая часть заполняется значениями знакового бита (сохраняет значение знаковой интерпретации x)

Битовые сдвиги:

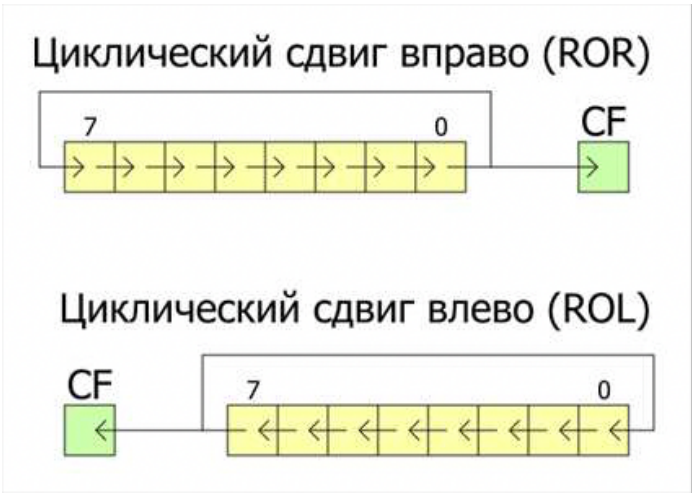
биты >> (величина сдвига)

- логические (при сдвиге вправо освободившийся разряд(старший) заполняется 0)
  101 -> 010  // аналог беззнакового деление на 2 с остатком
  В случае сдвига влево - инициализируется 0

- арифметические (при сдвиге вправо - заполняется копией знакового бита ) 1111 1011 -> 1111 1101 // знаковое  деление на 2
  В случае сдвига влево - инициализируется 0
  // аналог умножения на 2 (+ выполняется быстрее)

Остаток будет равен биту CF (вышедшему за разрядную сетку)

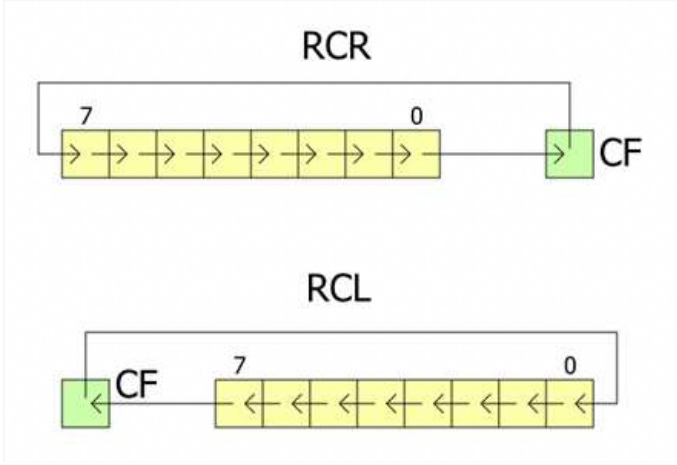
- циклические




// Перемещает элементы N-битной цепочки

Циклический сдвиг через флаг переноса
Ячейка инициализируется значением флага-переноса CF, а сама ячейка CF  замещается разрядом, выпавшим за разрядную сетку (результат зависит от текущего значения CF )

// перемещает элементы N+1 битной цепочки




## 7.Для чего нужно знать порядок следования байтов на вашем компьютере?

Прямой порядок - little endian
Обратный порядок -Big Endian

В позиционных системах счисления - согласно арабской традиции: старший - слева, младший - справа

Дамп памяти: младшие - слева, старшие справа

В Дампе памяти:
0x0A0B - нужно записать в память
0A - старший, 0B - младший
С прямым порядком: 0B0A
С обратным: 0A0B

x = 0x0A0B0C0D
(4 bytes)

в памяти
0A - k
0B - k+1
0C - k+2
0D - k+3

Обратный порядок (старший бит по младшему адресу)
0A0B0C0D

Прямой:
0D0C0B0A

Прямой порядок удобен при работе с числами большой разрядности с помощью процессора малой разрядности (можно обращаться к памяти последовательно).
Обратный принят в протоколе TCP/IP

В x86 используется прямой порядок байт