Компьютерные системы (мой конспект)

Список использованных источников:

1. *Computer Systems / Stanley J. Warford*
2. *Dive into systems / Suzanne J. Matthews, Tia Newhall, and Kevin C. Webb*
3. *Great ideas in computer architecture / видеокурс от университета Беркли*

Вводная

Что такое компьютерные системы? “Компьютерная система включает в себя аппаратное ***(hardware)*** и программное обеспечение ***(software)***, которые обеспечивают работу компьютера с пользователями и программами. В частности, компьютерная система состоит из следующих компонентов:

Порты ввода-вывода (I/O) позволяют компьютеру получать информацию из окружающей среды и отображать ее пользователю.

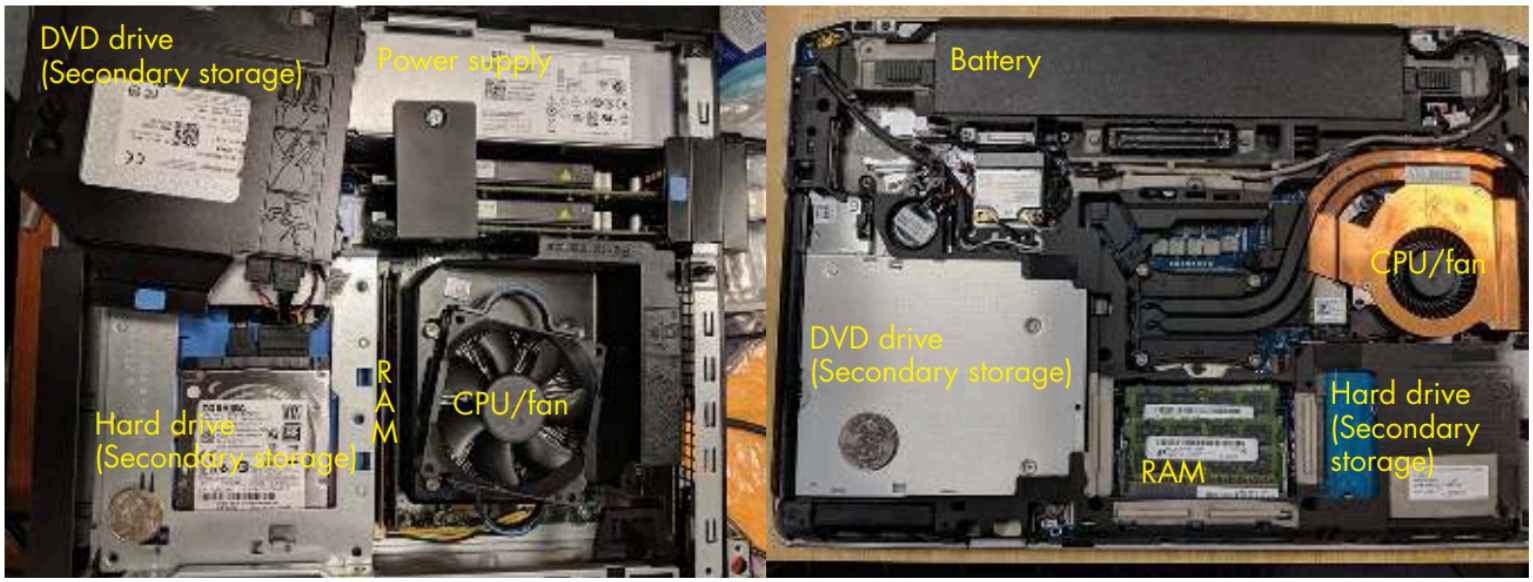
Центральный процессор (ЦП) выполняет инструкции и вычисляет данные и адреса памяти.

В памяти с произвольным доступом (RAM) хранятся данные и инструкции выполняемых программ. Данные и инструкции в оперативной памяти обычно теряются при отключении питания компьютерной системы.

Вторичные устройства хранения данных, например жесткие диски, хранят программы и данные даже при отсутствии питания компьютера.

Операционная система – это программный слой (software layer), находящийся между hardware и software, который пользователь запускает на компьютере. ОС реализует программные абстракции и интерфейсы, которые позволяют пользователям легко запускать и взаимодействовать с программами в системе. Она также управляет базовыми аппаратными ресурсами и контролирует, как и когда выполняются программы. В ОС реализованы абстракции, и механизмы, обеспечивающие эффективный, защищенный и бесперебойный одновременный запуск нескольких программ в системе”. (см. Dive into systems, pp. 8)

Эти компонеты можно увидеть ниже (слева – компьютер, справа – ноутбук):



Великие идеи в компьютерной архитектуре (см. лекции Беркли по компьютерной архитектуре):

1. *Абстракция;*
2. *Закон Мура;*
3. *Иерархия памяти;*
4. *Параллелизм;*
5. *Оценки и повышение производительности;*
6. *Надежность через резервирование.*

Всё есть 100101110100101!

Всякая информация хранится в двоичной системе счисления.

Бит – наименьшая единица информации, которую компьютер может обрабатывать и хранить. Бит всегда находится в одном из двух физических состояний – 1 или 0 **/** true или false и т.д.

Байт – организованный блок из 8 битов.

Как конвертировать двоичное число в десятичное:

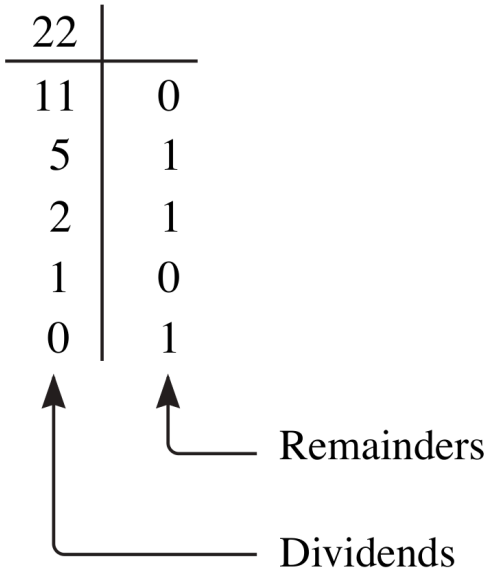
$$1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

(a) The binary number 10110.

$$5 \times 10^4 + 8 \times 10^3 + 0 \times 10^2 + 3 \times 10^1 + 6 \times 10^0$$

(b) The decimal number 58,036.

Как конвертировать десятичное число в двоичное (порядок по итогу будет обратный – 10110)



Старший бит (самый первый) – является признаком отрицательного или положительного числа (1 = отрицательное, 0 = положительное).

```
00001101  (decimal 13)
```

Next, “flip the bits” (change all zeros to ones, and vice versa):

```
11110010
```

Finally, adding one yields 0b11110011. Sure enough, applying the formula for interpreting a two’s complement bit sequence shows that the value is −13:

Арифметические операции с битами

Все операции – сложения, вычитания, умножения и деления работают с битами. Есть специальные регистры внутри ЦП, отвечающие за эти операции. Также внутри ЦП есть специальные биты (status bits), отвечающие за переполнение, переноса, и прочее. Ниже:

N – от слово *Negative* (отрицательное). Если число отрицательное, то устанавливается N = 1, в ином случае 0.

Z – от слово *Zero*. Если все биты (00000) являются нулями, то устанавливается Z = 1, в ином случае 0.

V – от слово *Overflow*. Отвечает за переполнение. Используется только для sign bit (1). Если сложите два положительных числа и получите отрицательное, то V = 1, аналогично если сложите два отрицательных числа и получите положительное, то V =1. Иное не возможно

C – от слово *Carry*. Отвечает за перенос числа.

...Еще шестнадцатеричная система счисления

Шестнадцатеричный счет по сути является аббревиатурой для двоичного счета

0	7	E	15	1C	23
1	8	F	16	1D	24
2	9	10	17	1E	25
3	A	11	18	1F	26
4	B	12	19	20	.
5	C	13	1A	21	.
6	D	14	1B	22	.

Введение в Си

C – это компилируемый, статически типизируемый язык программирования.

В языке программирования C есть три различных вида переменных – *глобальные*, *локальные* и *динамически аллоцируемые*. Значение переменной хранится в оперативной памяти компьютера, но где именно оно хранится, зависит от вида переменной. Существует три специальных раздела памяти, соответствующих трем видам переменных

Модель памяти C

1. *Глобальные переменные хранятся в фиксированном месте в памяти;*
2. *Локальные переменные и параметры хранятся в runtime stack;*
3. *Динамически аллоцируемая переменная хранится в куче;*

Немного про runtime stack:

Стек – это контейнер значений, который хранит значения с помощью операции push и извлекает их с помощью операции pop. Каждый выполняемый оператор языка Си является частью функции. Функция Си имеет тип возврата, имя и список параметров. Программа состоит из специальной функции main, которая является функцией, вызываемой операционной системой. Программа выполняется путем выполнения утверждений в функции main. Оператор main может вызывать другую функцию.

Все они – runtime stack, куча, фиксированная память находятся в памяти компьютера.

Что происходит в runtime stack при вызове функции?

- *push хранилище для возвращаемых значений;*
- *push актуальные параметры;*
- *push возвращаемый адрес.*

Что возвращает функция?

- *pop локальные переменные*
- *pop возвращаемый адрес и используй его для определения следующей инструкции для выполнения.*
- *pop параметры*
- *pop возвращаемое значение*

Хранилище для возвращаемых значений – означает, что вызывающая функция резервирует пространство в стеке для хранения возвращаемого значения перед вызовом функции. Этот процесс важен для правильного управления стеком вызовов и обеспечения корректного возврата результатов выполнения функции.

Есть два вида параметров. Формальные (*formal*) и актуальные (*actual*) параметры. **Формальные параметры** — это параметры, которые объявляются в определении функции, метода или процедуры. Они служат в качестве местозаполнителей для значений, которые будут переданы функции при ее вызове. Формальные параметры определяются в заголовке функции и не имеют конкретных значений до тех пор, пока функция не будет вызвана.
В нашем примере `void printBar(int n)` – n является формальным параметром.
Актуальные параметры — это конкретные значения, которые передаются функции при ее вызове. Эти значения соответствуют формальным параметрам по порядку или по имени. Актуальные параметры предоставляют данные, с которыми функция будет работать.

Возвращаемый адрес – это адрес в памяти, куда программа должна вернуться после завершения вызова функции. Это ключевая концепция в управлении вызовами функций и возвратом в исходную точку программы. Пример:

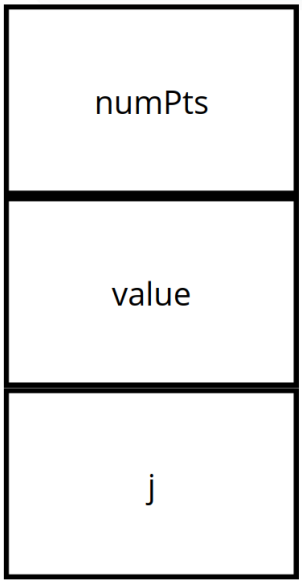
```
void functionB() {
    // Код функции B
}
void functionA() {
    functionB(); // Вызов функции B
    // Адрес возврата будет здесь
}
int main() {
    functionA(); // Вызов функции A
    // Адрес возврата будет здесь
    return 0;
}
```

- Когда `functionA()` вызывается из main, адрес возврата указывает на строку после вызова `functionA()`.
- Когда `functionB()` вызывается из `functionA()`, адрес возврата указывает на строку после вызова `functionB()`.
- Когда `functionB()` завершает выполнение, она возвращается к адресу возврата в `functionA()`.
- Когда `functionA()` завершает выполнение, она возвращается к адресу возврата в main.

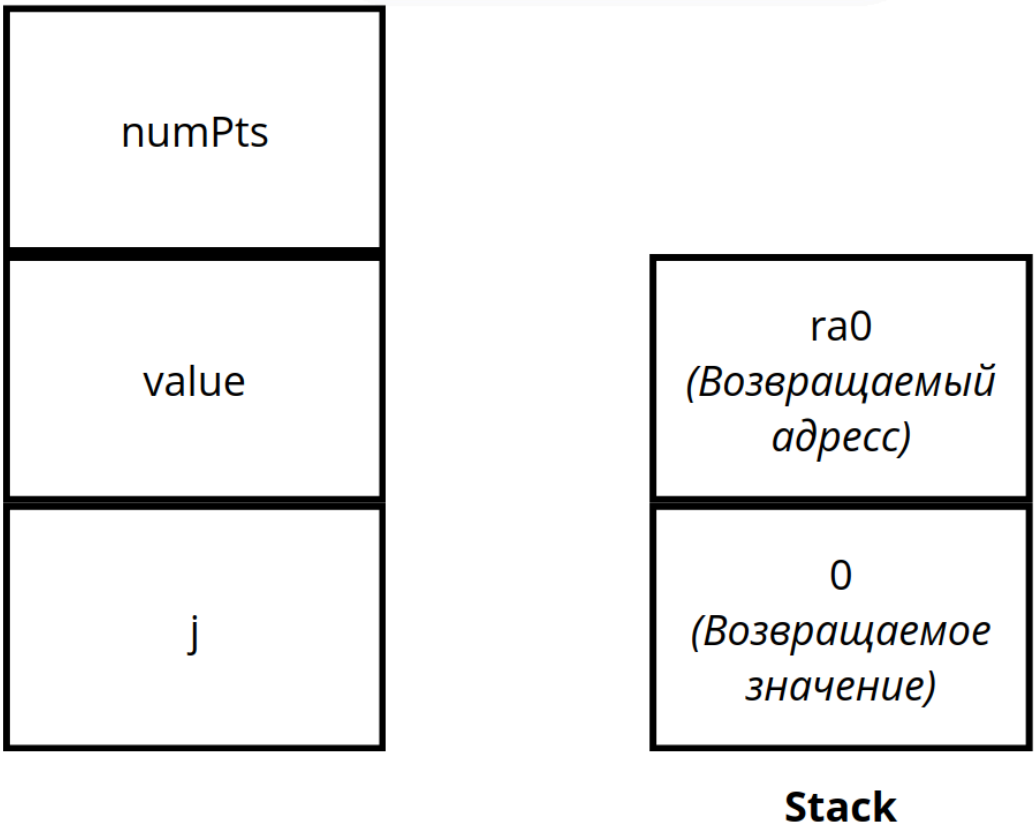
Для практической иллюстрации вышесказанного:

```c
#include <stdio.h>

//Глобальные переменные объявляются вне функций
int numPts;
int value;
int j;

void printBar(int n) {

   int k;

   for (k = 1; k <= n; k++) {

      printf("*");

   }

   printf("\n");

}

int main() {

   scanf("%d", &numPts);

   for (j = 1; j <= numPts; j++) {

      scanf("%d", &value);

      printBar(value); //printBar - возвращает адрес возврата

   }

   return 0; //Возвращаемое значение

}
```

Три переменные **numPts**, **value** и **j** являются глобальными, ибо объявлены вне функций, следовательно будут храниться в фиксированном месте в памяти.



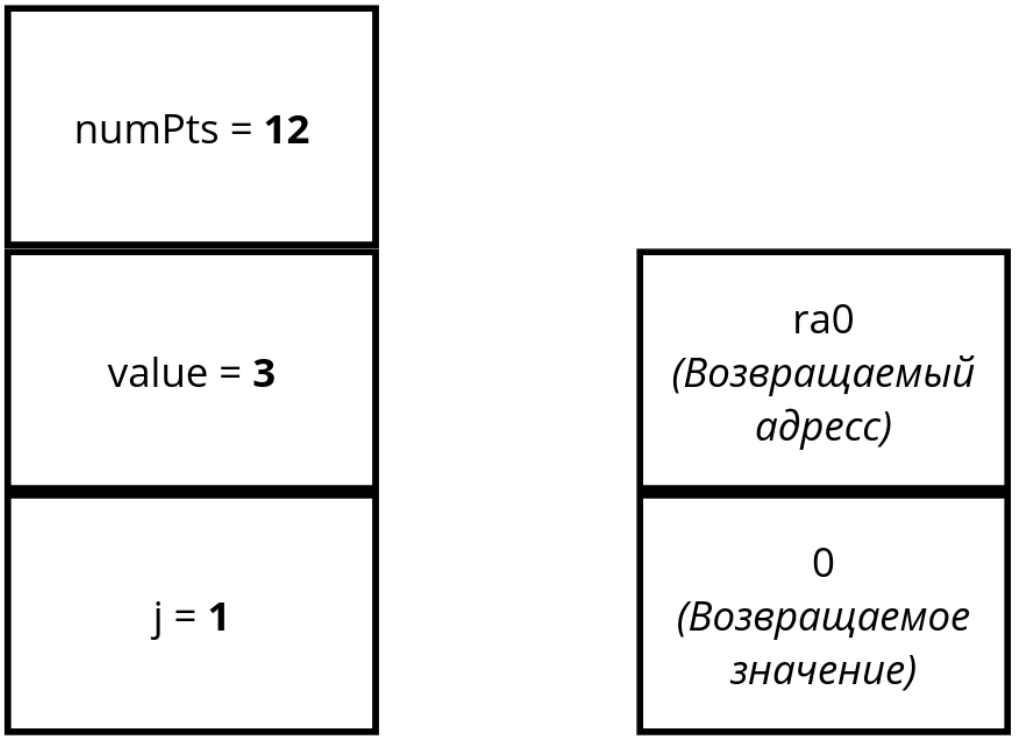
Функция **`main()`** является главной функцией, которую запускает операционная система при старте программы. После выполнения **`main()`** операционная система получает *возвращаемое значение* (0), что означает успешное завершение программы, и завершает выполнение программы.



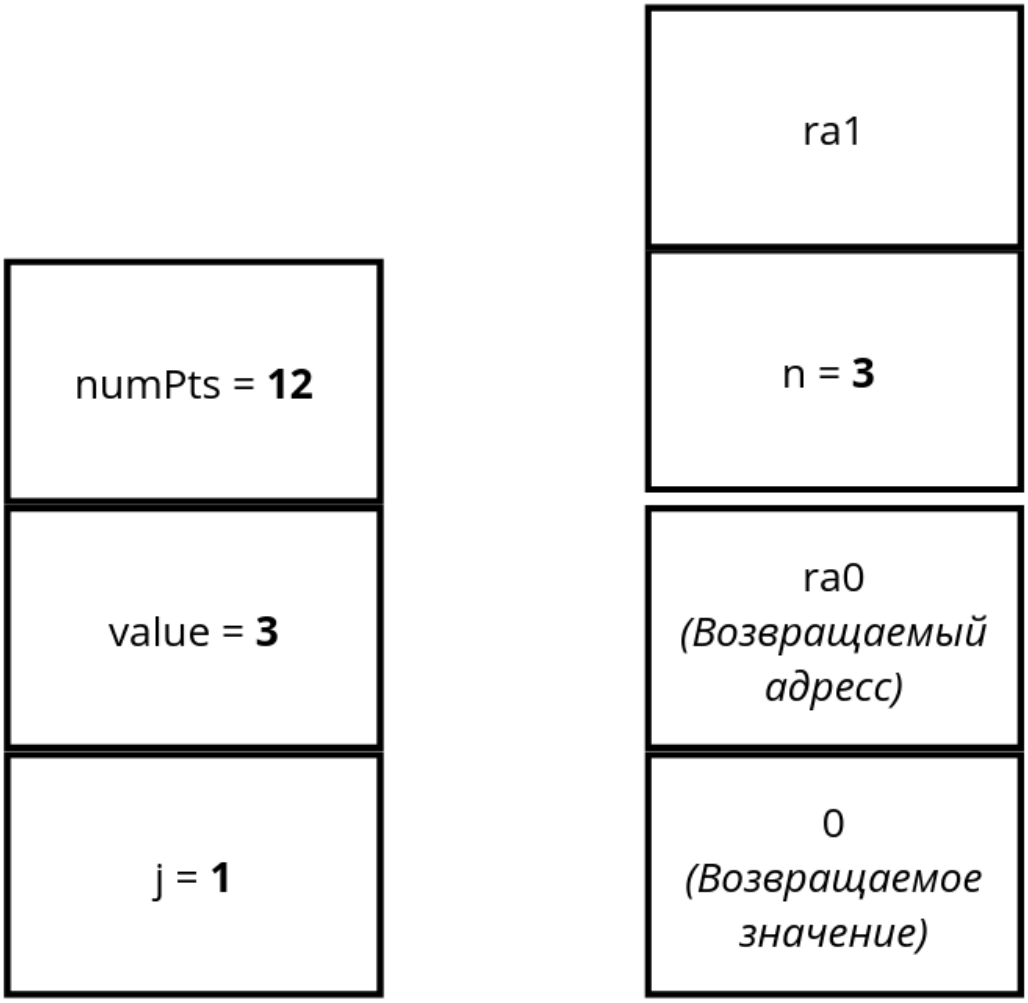
На рисунке в качестве адреса возврата указано значение ra0, которое является адресом инструкции в операционной системе, которая будет выполнена при завершении программы.

`scanf("%d", &numPts)` – из I/O пользователь вводит число 12
`for (j = 1; j ⩽ numPts; j++)` – переменная получает значение 1

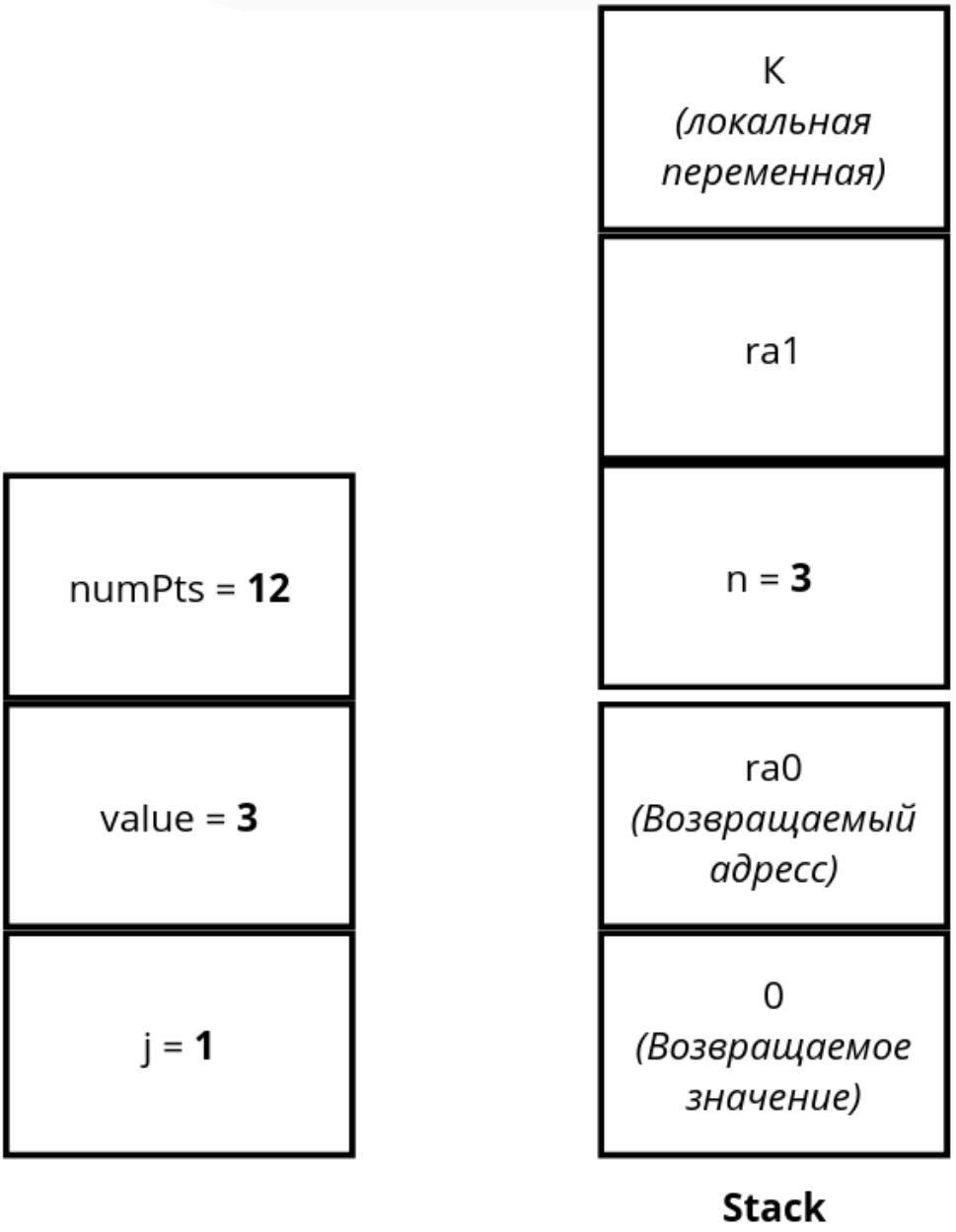
`scanf("%d", &value);` – из I/O Пользователь вводит число 3



`printBar(value)` – функция получает значение значение из value и возвращает адрес (ra1)



`int k;` – является локальной переменной в функции `void printBar(int n),` для которой выделяется хранилище в памяти.

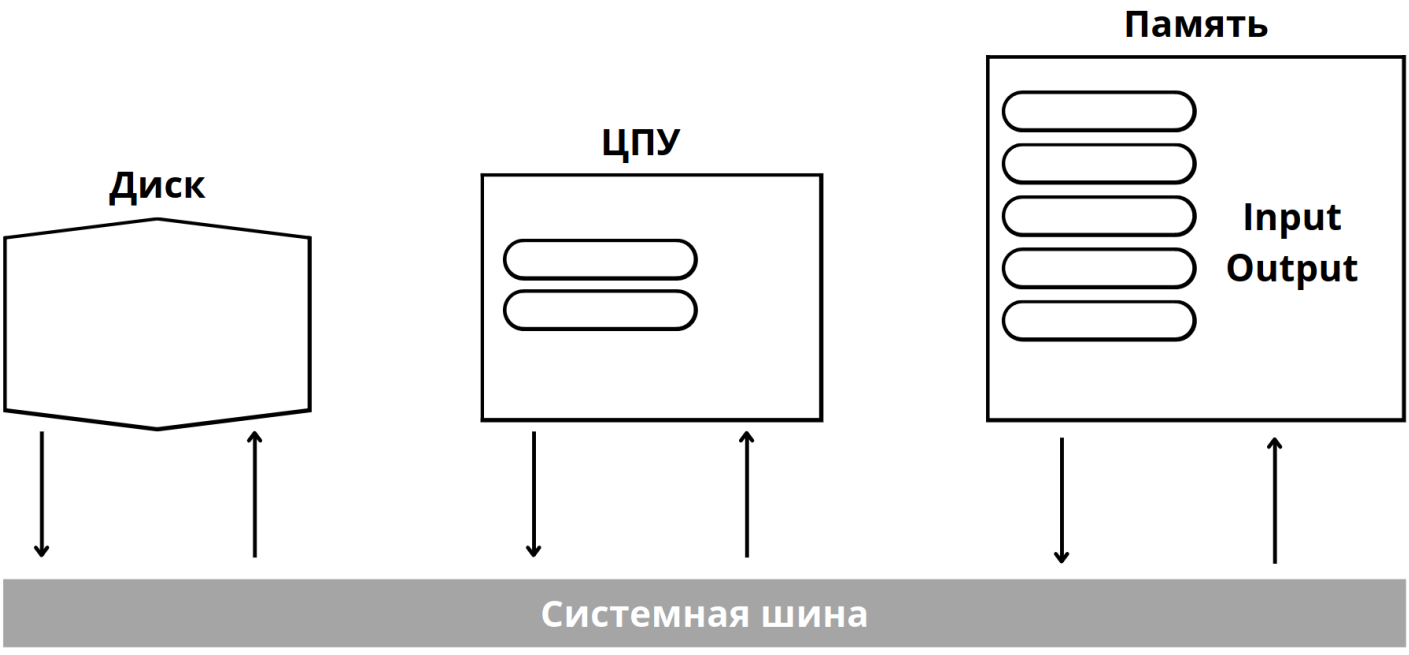


Динамическое распределение памяти:

В языке программирования C (и многих других), куча (heap) — это область динамической памяти, которая используется для хранения объектов, созданных во время выполнения программы. Управление этой памятью осуществляется с помощью функций, таких как malloc, calloc, realloc и free.

Что попадает в кучу?

- **Массивы**: Массивы переменной длины, размер которых определяется во время выполнения.
- **Структуры**: Сложные структуры данных, которые могут содержать другие структуры или массивы.
- **Строки**: Строки переменной длины, которые создаются и изменяются во время выполнения программы.


Память
Input
Output
ЦПУ
Диск
Системная шина