ARM Microcontroller Internship Programme

Trainer: *Kiran Jaybhave*

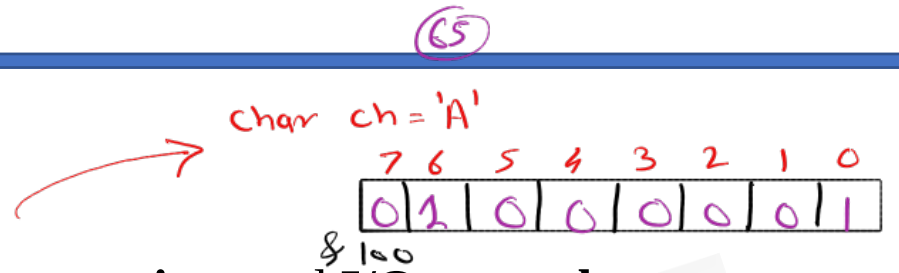
Sunbeam – Industrial Training Programme

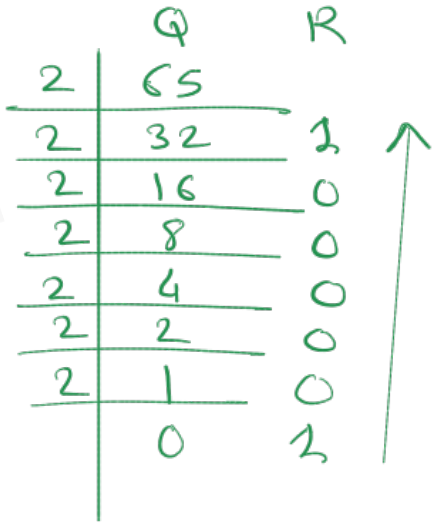
Sunbeam Pune (Hinjawadi)

- **Prerequisites**
  - **C Programming:**
    - Proficiency in C syntax, pointers, memory management, structures, bit manipulation, and preprocessor directives
  - **Basic Electronics:**
    - Understanding of digital electronics, analog circuits, voltage levels, timing diagrams, and basic circuit analysis

Bit Manipulation

- **What is Bit Manipulation?**
  - Operations performed directly on **bits** (0s and 1s).
  - Used heavily in **embedded systems**, **register programming**, and **I/O control**.





- **Why Bit Manipulation is Important?**
  - Used for **GPIO control**, **interrupt flags**, **status registers**.
  - Essential for **memory-mapped I/O** and **hardware-level programming**.
  - Helps reduce code size and improves speed.

- # Common Bit Operations

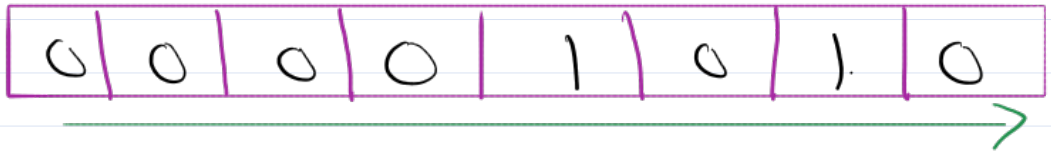
1. **Set a bit** → Make a bit = 1
   value |= (1 << n);
2. **Clear a bit** → Make a bit = 0
   value &= ~(1 << n);
3. **Toggle a bit** → Flip bit (0→1 or 1→0)
   value ^= (1 << n);
4. **Check a bit** → Test if bit = 1
   if (value & (1 << n))
5. **Masking** → Isolate selected bits
   value & mask
6. **Shifting** → Move bits left or right
   value << n, value >> n

Print Num in Binary

-10 ⇒ True
10 ⇒ True

0 ⇒ False

num = 10

num ⇒ 

& 100

mask = 0 0 0 0 1 0 0 0
VP 0 0 0 0 1 0 1 0
&
0 0 0 0 0 0 0 0 ⇒ 0
0 0 0 0 0 0 0 0 ⇒ 0
0 0 0 0 0 0 0 0 ⇒ 0
0 0 0 0 0 0 0 0 ⇒ 0
0 0 0 0 1 0 0 0 ⇒ 8 ⇒ 1

```
void print_bin ( void *vp , int size)
{
    mask = 0x 80;
    while ( mask )
    {
        if ( mask & *(char*)vp )
            pf (1)
        else
            pf (0)
        mask = mask >> 1;
    }
}
```

```
void *vp = 100;
size     = 1;
```

⇒

```
char mask = 0x 80;
```

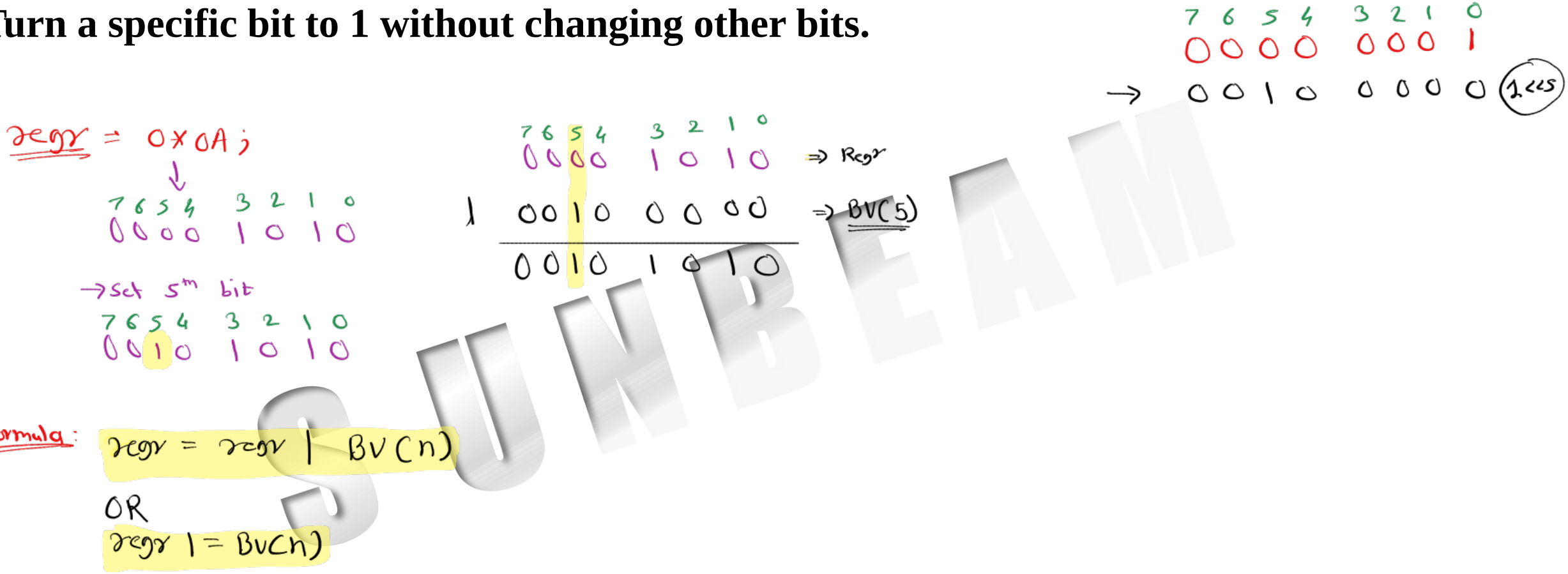
int regr = 0XFAFA;

00	00	FA	FA
0000 0000	0000 0000	1111 1010	1111 1010
mask = 0X80	mask = 0X80	mask = 0X80	mask = 0X80
103	102	101	100

Set a bit

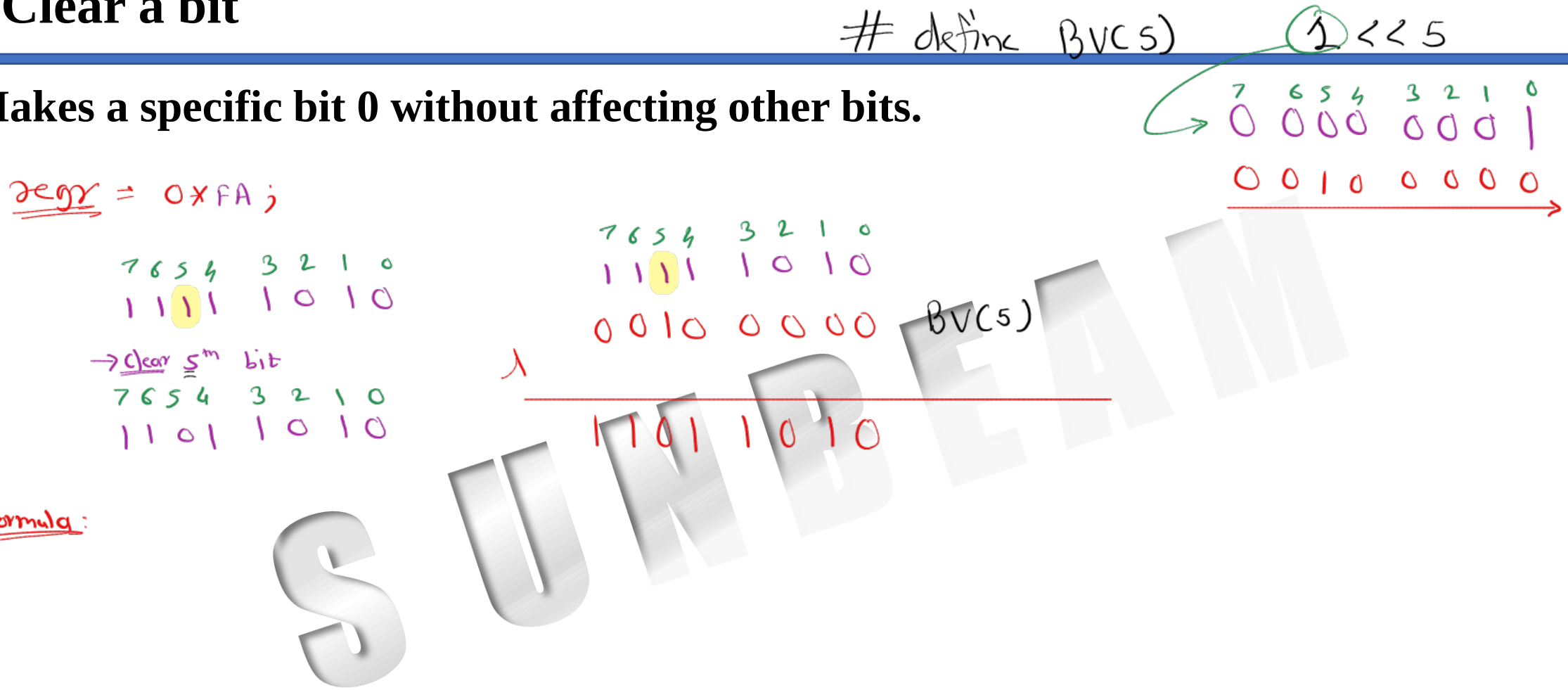
#define BV(n) 1<<n

- **Turn a specific bit to 1 without changing other bits.**



➢ Clear a bit

- **Makes a specific bit 0 without affecting other bits.**



➢ Toggle a bit

- **Flips the bit value (0 becomes 1, and 1 becomes 0).**

➢ Check a bit

- **Tests whether a specific bit is 1 or 0.**

➢ Masking

- **Extracts only the required bits and hides the rest.**

➢ Left Shifting

- **Moves bits to the left, multiplying the value by powers of two.**

➢ Right Shifting

- **Moves bits to the right, dividing the value by powers of two.**

➢ Read value from bit n^{th} to n^{th} of register

➢ Write value from bit n^{th} to n^{th} of register

Thank you!

Kiran Jaybhave

email – kiran.jaybhave@sunbeaminfo.com

32 bit → 4
64 bit → 8

data type	variable	pointer
int	n1 (4)	*p (4)
char	c1 (1)	*p (4)
double	d1 (8)	*p (4)

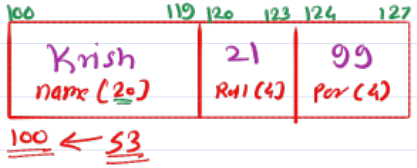
```
struct student
{
    char name[20];
    int Roll;
    float Per;
} s1, *p;
```

→ size = 28

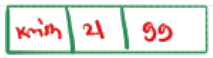
s1 → 28
p → 4

Function

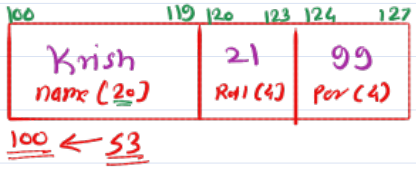
Pass by value




```
int main()
{
    strat student S3;   // (size => 28)

    Strat studen *p ; // size→4

    print_student ( S3 ) // Pass by Valuc

}

void print_student (struct student S4)
{
    pf( S4.name , S4.Roll , S4.per )

}
```



Pass by add




```
int main()
{
    strat student S3;   // (size => 28)

    Strat studen *p ; // size→4

    print_student ( & S3 ) // Pass by Valuc

}

void print_student (struct student *S4)
{
    pf( S4→name , S4→Roll , S4→per )

}
```

& 100

address

4byte

```
int ar[3] = { 10, 20, 30 }
```

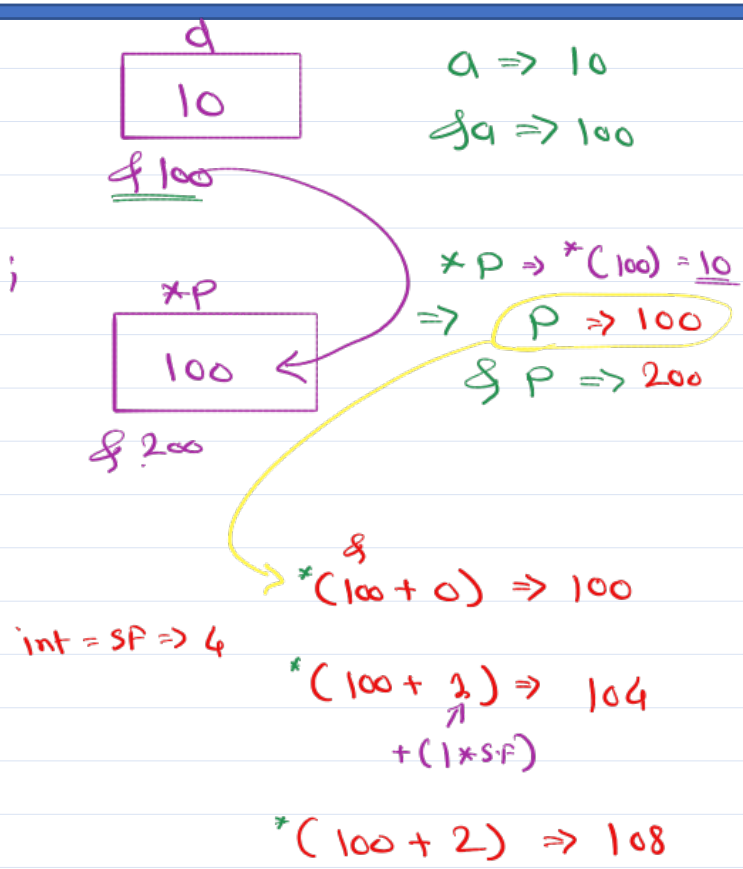
0 1 2

& 100 104 108

12 bytes

ar[0] => 10	*(ar+0) => 10
ar[1] => 20	*(ar+1) => 20
ar[2] => 30	*(ar+2) => 30

```
int a = 10;
int *p = &a;
```



At the time Read value we need to type cast.

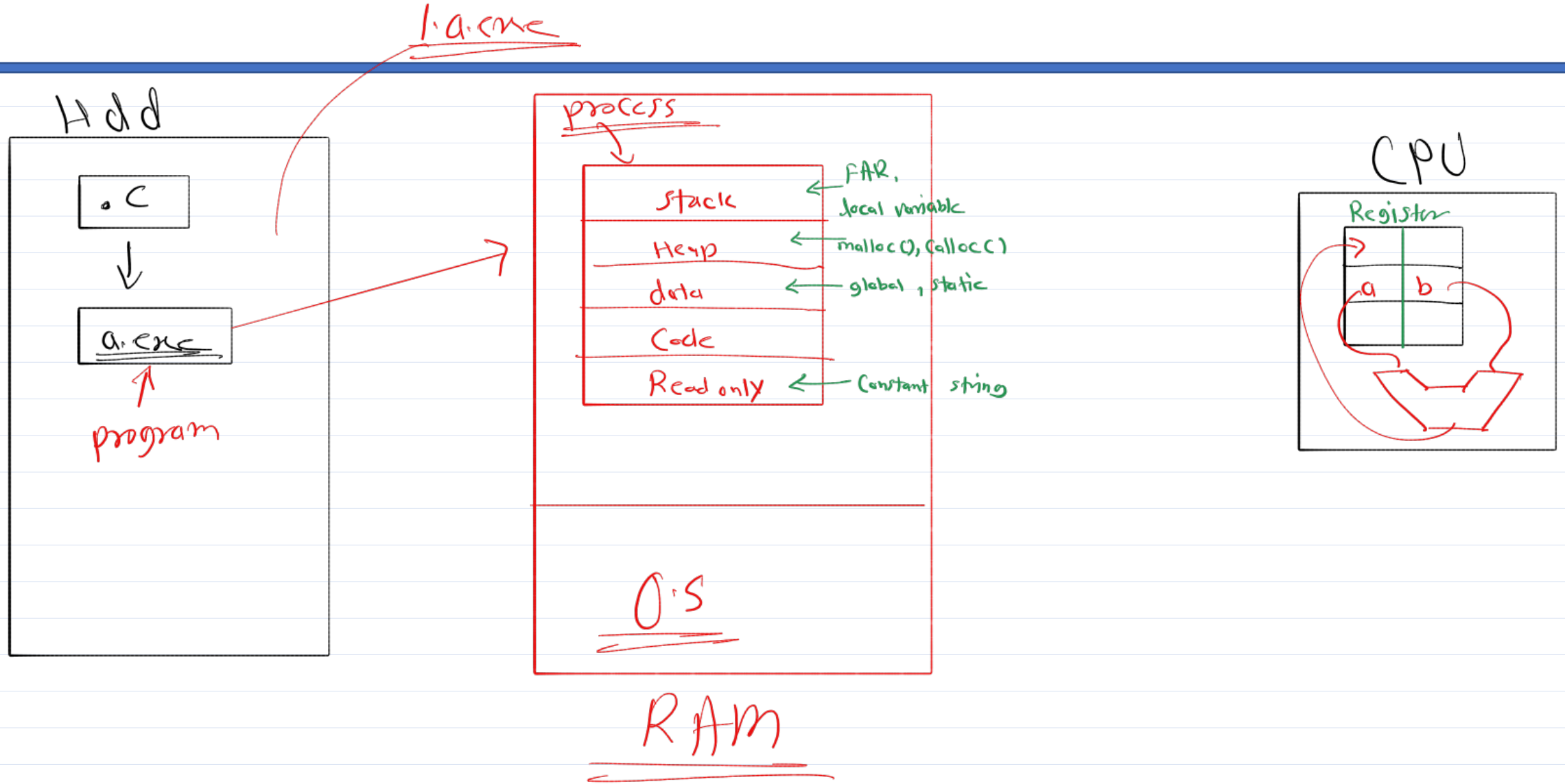
```
int    *p ;
char   *cp ;
double *dp ;
struct student *sp ;
```

```
void *vp ;
```

Generic pointer

Store address of any data type variable.

```
void *vp = p    =>  *(int *)vp
void *vp = cp   =>  *(char *)vp
void *vp => dp  =>  *(double *)vp
void *vp => sp
```


1.a.exe
Hdd
.c
a.exe
program
process
stack
FAR, local variable
Heap
malloc(), calloc()
data
global, static
Code
Read only
Constant string
O.S
RAM
CPU
Register
a
b