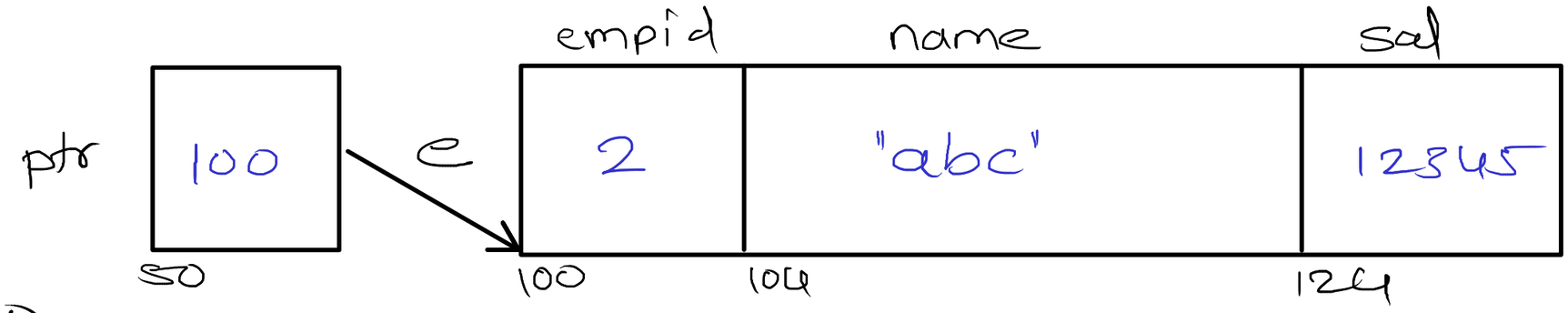
```
struct emp {
    int empid;
    char name[20];
    float sal;
} e;
```

struct emp * ptr = &e;



&e = 100

&e.empid = 100

&e.name = 104

&e.sal = 124

(char *) &e.empid − (char *) &e

$= \frac{100 - 100}{1} = 0$

(char *) &e.name − (char *) &e

$= \frac{104 - 100}{1} = 4$

(char *) &e.sal − (char *) &e

$= \frac{124 - 100}{1} = 24$

&e = 0

&e.empid = 0

&e.name = 4

&e.sal = 24

(char *) &e.empid − (char *) &e

$= \frac{0 - 0}{1} = 0$

(char *) &e.name − (char *) &e

$= \frac{4 - 0}{1} = 4$

(char *) &e.sal − (char *) &e

$= \frac{24 - 0}{1} = 24$

```
unsigned char num = 125;

while(mask)
{
    if ( num & mask)
        pf("1");
    else
        pf("0");
}
```

MSB LSB
(0 1 1 1 1 1 0 1)

& 1 0 0 0 0 0 0 0

0

MSB LSB
(0 1 1 1 1 1 0 1)

& 0 1 0 0 0 0 0 0

1

MSB LSB
(0 1 1 1 1 1 0 1)

& 0 0 1 0 0 0 0 0

1

mask = 1000 0000
0100 0000
0010 0000
0001 0000
0000 1000
0000 0100
0000 0010
0000 0001
0000 0000

signed char num = 125;

MSB LSB
(0 1 1 1 1 1 0 1)

num << 3 1110 1000

MSB LSB
(0 1 1 1 1 1 0 1)

num >> 3 0000 1111

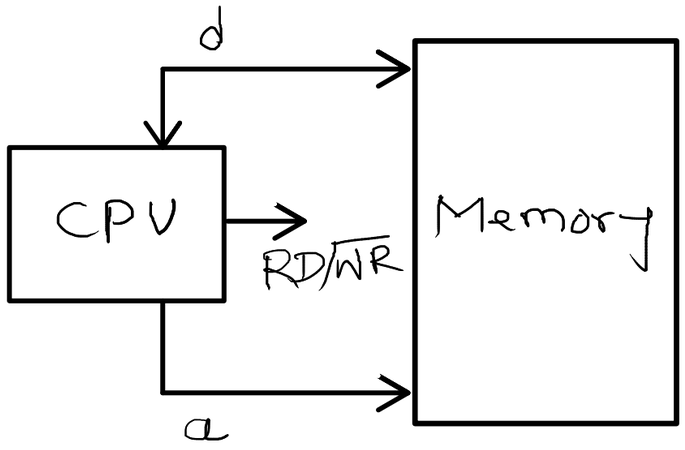
signed char num = -128

(1000 0000)

num << 3 0000 0000

(1000 0000)

num >> 3 1111 0000



ARM - word = 4 bytes
hword = 2 bytes

x86 - word = 2 bytes
dword = 4 bytes

Word Unaligned Access

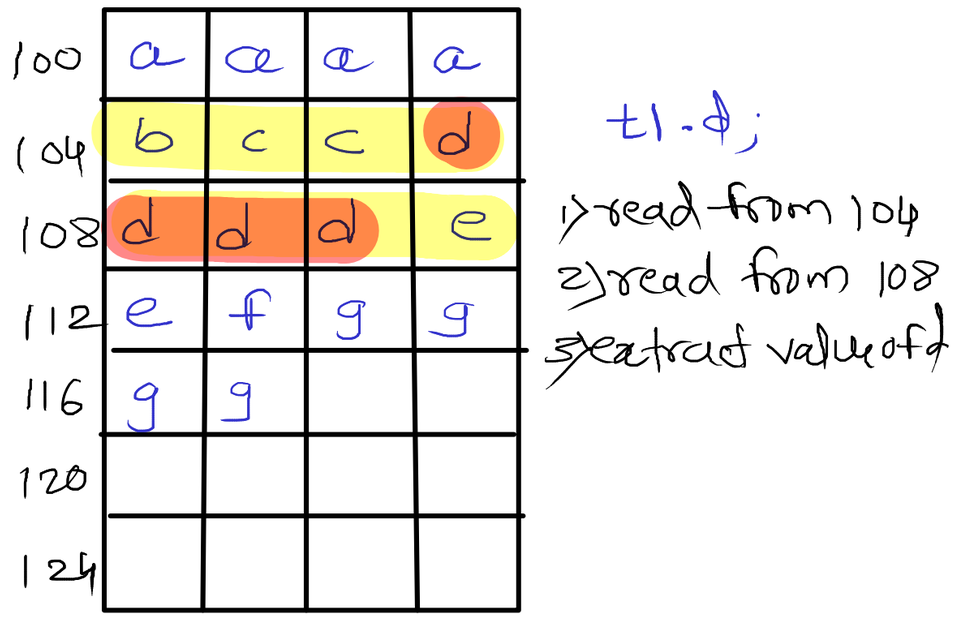
CPU can read/write any size of data on any address

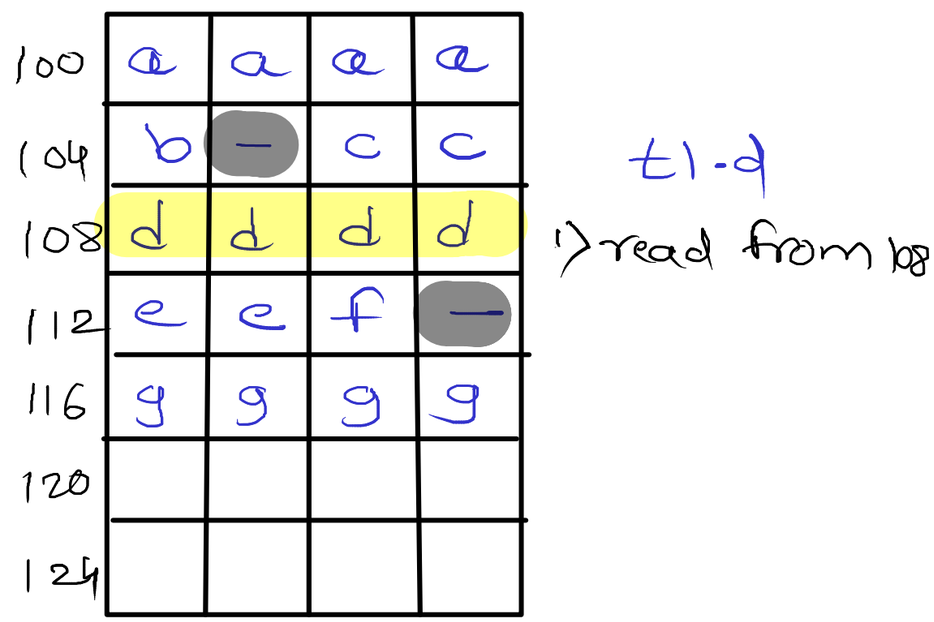
Word Aligned Access – efficient

CPU can read/write the data of size from address which is multiple of size

char - size = 1, address - multiple of 1
short - size = 2, address - multiple of 2
int - size = 4, address - multiple of 4

```
struct test {
    int a;
    char b;
    short c;
    int d;
    short e;
    char f;
    int g;
} t1;
```





Function calling conventions

1. How arguments are pushed on stack
   left to right / right to left
2. Who does the stack cleanup?
   called function / calling funtion

```
void fun(void)
{
    // To Do
}

int main(void)
{
    fun();
    return 0;
}
```

Called function

Calling function

Types of conventions:
1. Pascal
2. cdecl (C Lang)
3. stdcall

Pascal	left to right	called
cdecl	right to left	calling
stdcall	right to left	called

Pascal

```
void fun(int num1, int num2, int num3)
{
        // To Do
        // pop 30
        // pop 20
        // pop 10
        // return
}

int main(void)
{
        int n1 = 10, n2 = 20, n3 = 30;
        fun(n1, n2, n3);
        // push 10
        // push 20
        // push 30
        // call fun
        return 0;
}
```

Push Sequence : left to right
Stack cleanup : called function

cdecl

```
void fun(int num1, int num2, int num3)
{
	// To Do
	// return
}

int main(void)
{
	int n1 = 10, n2 = 20, n3 = 30;
	fun(n1, n2, n3);
	// push 30
	// push 20
	// push 10
	// call fun
	// pop 10
	// pop 20
	// pop 30
	return 0;
}
```

Push Sequence : right to left
Stack cleanup : calling function

stdcall

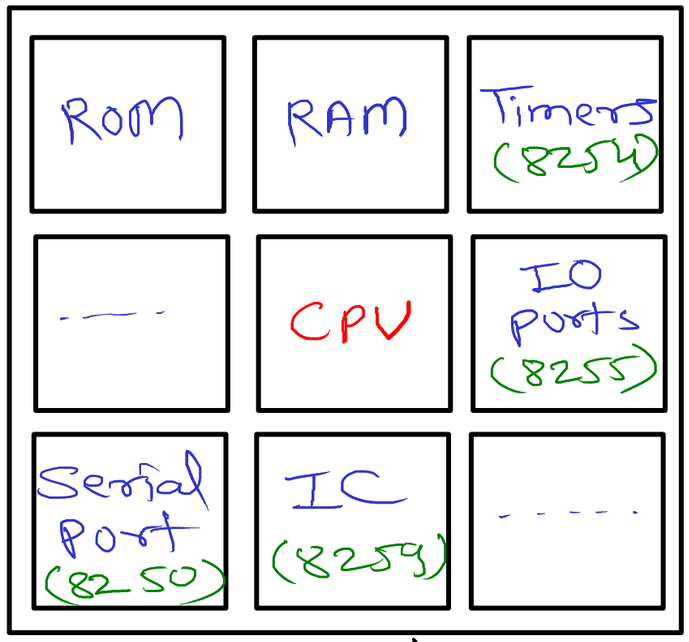
```
void fun(int num1, int num2, int num3)
{
	// To Do
	// pop 10
	// pop 20
	// pop 30
	// return
}

int main(void)
{
	int n1 = 10, n2 = 20, n3 = 30;
	fun(n1, n2, n3);
	// push 30
	// push 20
	// push 10
	// call fun
	return 0;
}
```

Push Sequence : right to left
Stack cleanup : called function

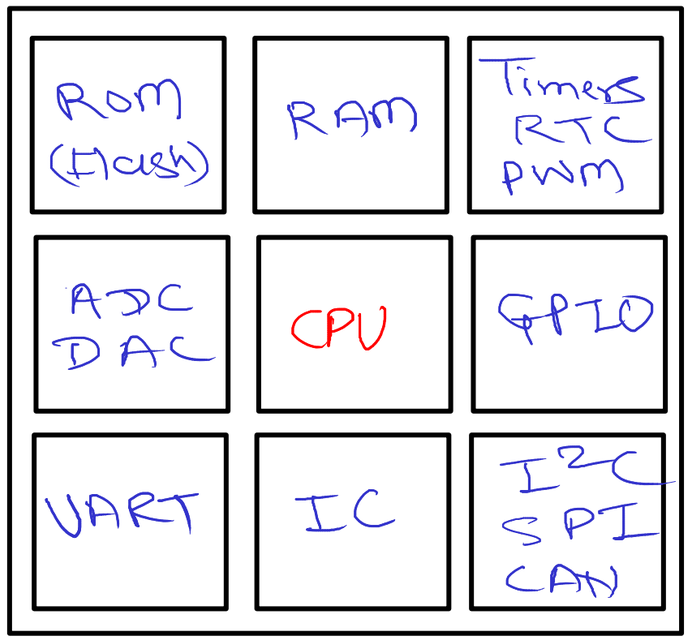
Micro Processor Vs Micro Controller

µP = ALU + Registers + EU + BIU

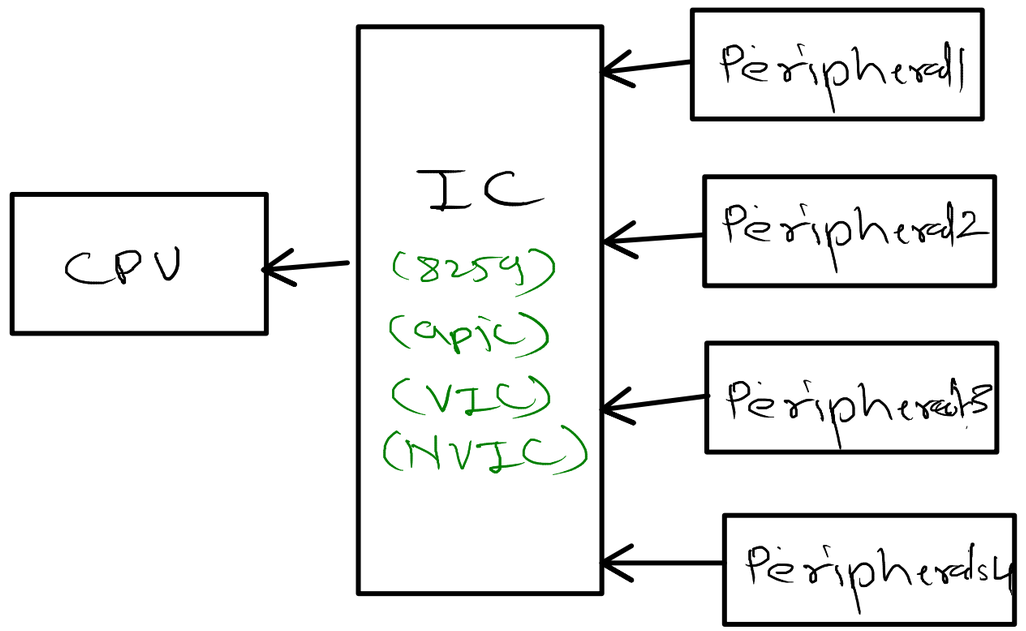


motherboard

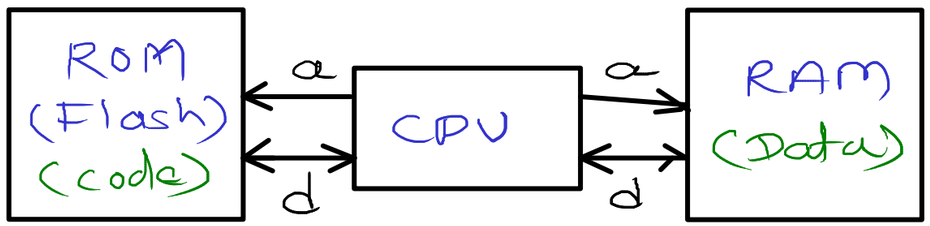
µC = ALU + Registers + EU + BIU + peripherals

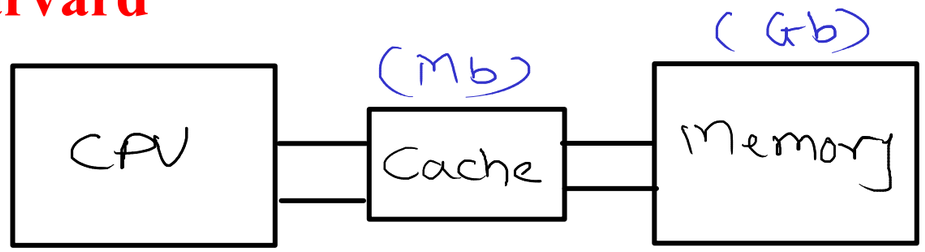


single chip (SOC)

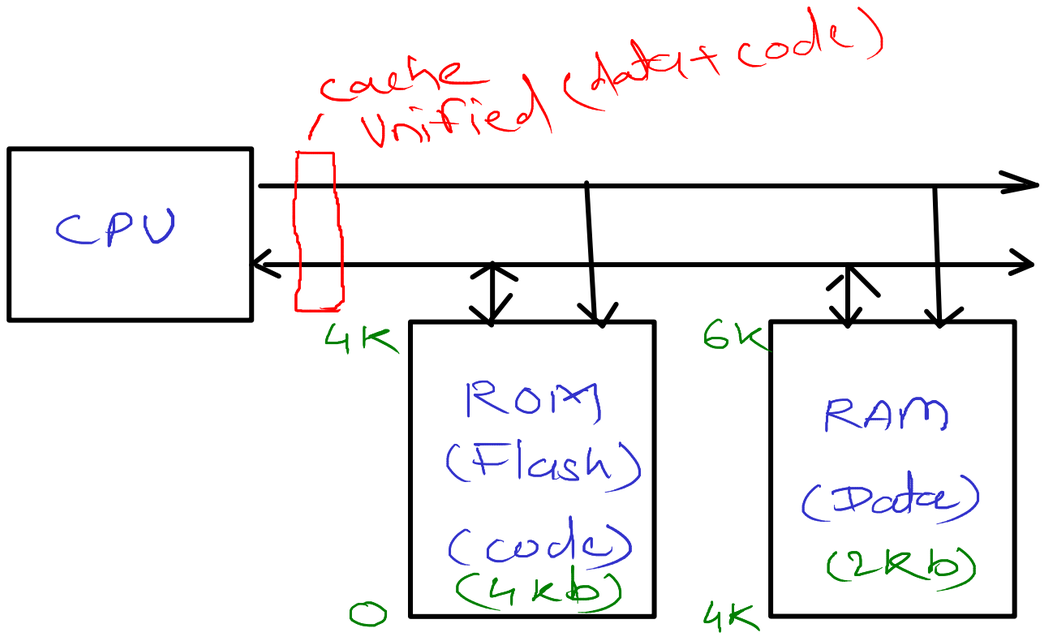


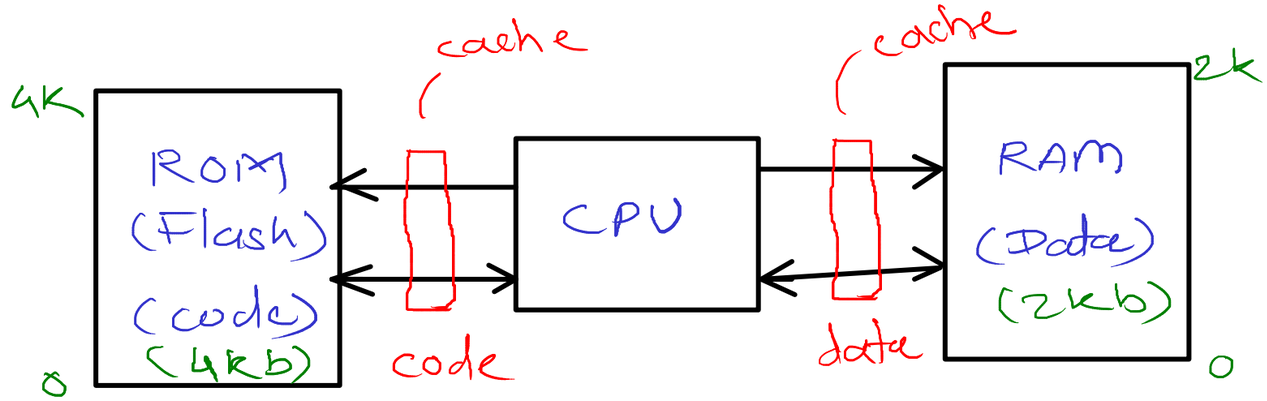
Von neumann Vs Harvard





- Associative access
  (key-address, value-data)
- Faster memories
- Hit-faster, Miss-slower
- older data is overwritten by new data if full





Super Harvard

Read only data (string constant) is kept with code inside ROMb