Structure (P.3)

⟹ Aligned & un-Aligned data access of struct:

```
struct data
{
    unsigned char d;
} X;
```

1B

→ 1 byte

```
struct data
{
    unsigned char d1;
    int d2;
} X;
```

5 byter

XX

→ 8 bytes (?)

⟹ Normal Size bound

char ⟶ 1B

int ⟶ 4 bytes , - ——-

Compiler ⟶ more efficient

↓

Performance ↑ ∝ $\frac{1}{\text{ex-time}\downarrow}$

EX-time ↓ ⟶ # instr ↓

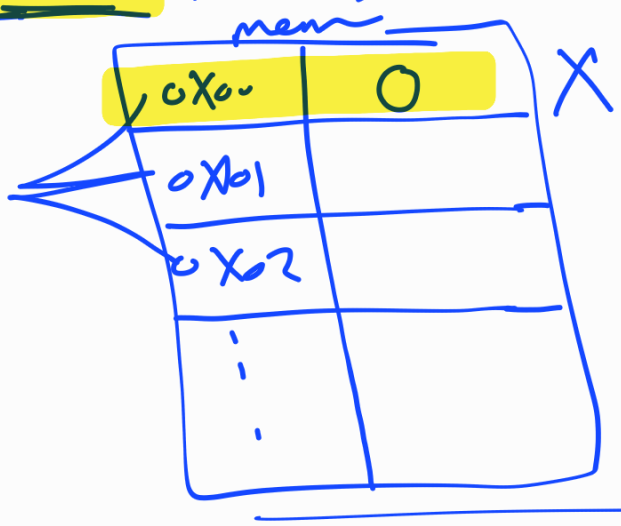
(Assembly)

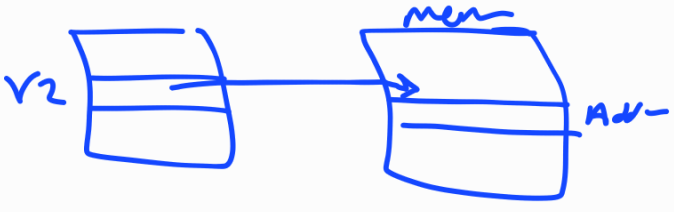
.text → Code

ARM Assembly

char X = 0; → Strb r2, [r3, #0]

Value, Addr, offset, Cast





word → 4 b

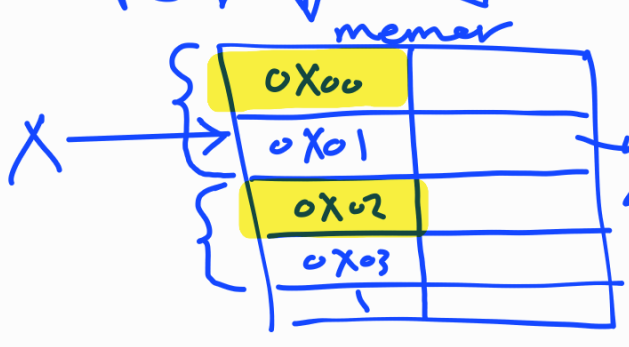
half word → 2 b

b

short X = 1; → strh r2, [r3, #0]

{strb ~ ~ ;
strb ~ ~ ~ }

Perf ↓



⇒ alligned

0x00

0x02

0x04 ....

int → 0x00 , 0x04, 0x08, ----

↳ 4 bytes

⇒ EX-

"Enum"

→ user defined datatype

eg

enum flag { Const1 (0), Const2 (1), ---- ConstN };

By default → 0 1 2 3 ----

enum week { Sat = 1, Sun = 2, Mon = 3 ---- };

Code ⟶ more professional (Readable)

⟹ enum ⟹ Space (int) = 4 byte

EX=

enum boolean { false (0), true (1) } check;

→ Code

if(check == true) { }

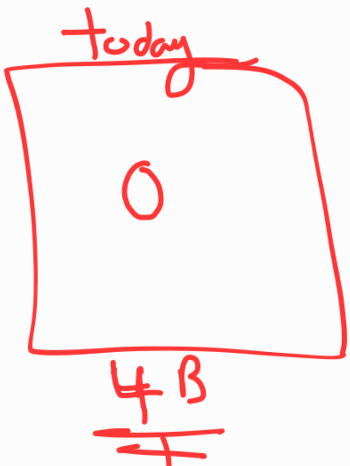
Embedded

es timer → modes ✓

enum → 00 0
→ 01 1
→ 10 2
→ 11 3

Ex

```
enum week { sat , sun , mon , - - - - };
             0     1     2
int main ()
 {
   enum week today;
                0
   today = Sat ;
 }
```



(one Value)

---

hint

```
enum week { sat = 0 , Sun = 1 , Mon = 4 , wed ,
            thu = -1 , fri }
```

wed → 5 ✓

fri → 2 ✓

⟹ All enum constants must be unique in their scope

ex enum state { working, failed };

~ result { failed, passed }; → error

---

sizeof enum ⟹ const int = 4 B

if

enum state { working = 1000 000 00000

, failed = 0 };

↳ 8 bytes #