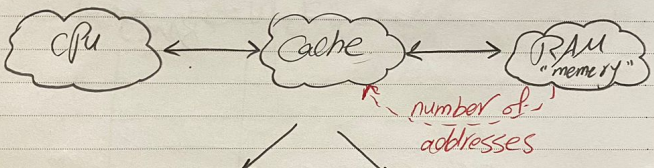
"Video 5"

MCu Architecture

Cache memory

* Static RAM (SRAM)
* high cost

*between CPU and memory.



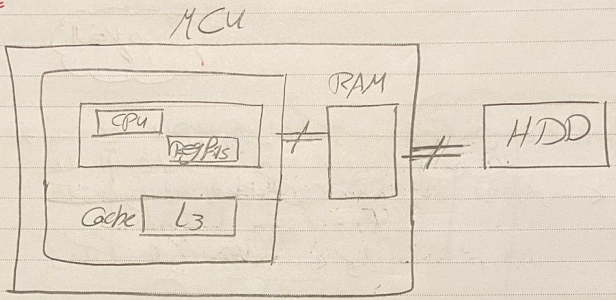
$$\text{hit ratio} = \frac{\text{no of hit}}{\text{total no of (hit+miss)}}$$

Cache ⟹ عبارة عن Cache memory + microcontroller

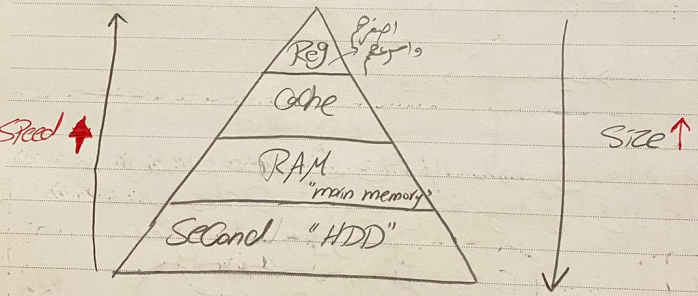
Cache memory: مخزن مؤقت مجموعة addresses من

microcontroller: المسؤول عن انه يجيب addresses الـ RAM to Cache memory

#



السرعة مع حجم الحجم



/ /

* Differance between

FPU , MPU , MMU

① FPU → floating Point unit

مسؤول عن العمليات على أرقام

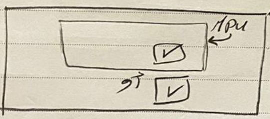
float ← datatype مثل

مثل GPU و DSP شغالة قبل كدا

*ممكن يكون جوا ال MPU أو برا ولكنه في الحالتين هو جوا MCU

حسب الحجم

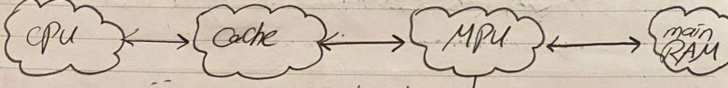
المفتقر



improve Performance.

---

② MPU → memory Protection unit – between cache و main memory

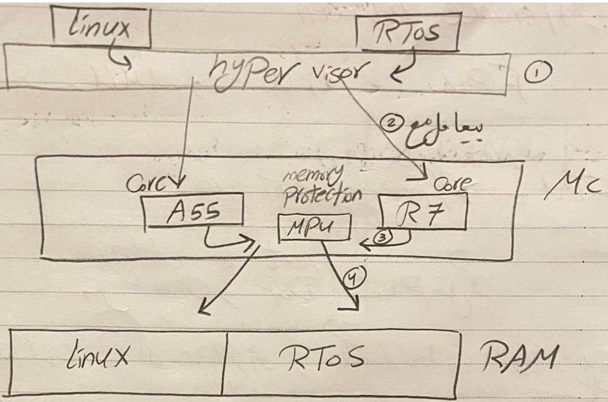


جواها registers مسؤولة عن تقسيم ال RAM

مجموعة Ranges

طب إيه الاستفادة ؟

مثال



لما يكون
عندي نظامين linux و RTos وكل واحد ليه
جزء ال Range of addresses من ال RAM "memory"

① رحت ال linux وال RTOS فيه عنصر اسمه
hyper visor لما بيجيلو أمر من linux أو RTos

② لو أنا عرفت أن MPU لما يكلمها R7 هتروح جزء RTos
ولو
كلمها A55 Core هتروح جزء linux
"Range of addresses
at linux"

③ لو بعت address الغلط هيبعتلي Bus fould
هيوقف ال system
فيه حاجة غلط

→ visual addresses

③ MMU → memory manage unit

لو عندي ٤ البرنامج المفروض يشتغل في 8G وأنا مساحتي 1G

هيشوف أنه شغال في 8G وهو الموجود عنده 1G
البرنامج

أهم حاجتين في embedded linux

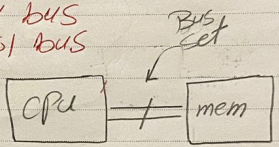
① هي سعر ال core
② وجود ال MMU في الشغل
أنه موجود في عوري

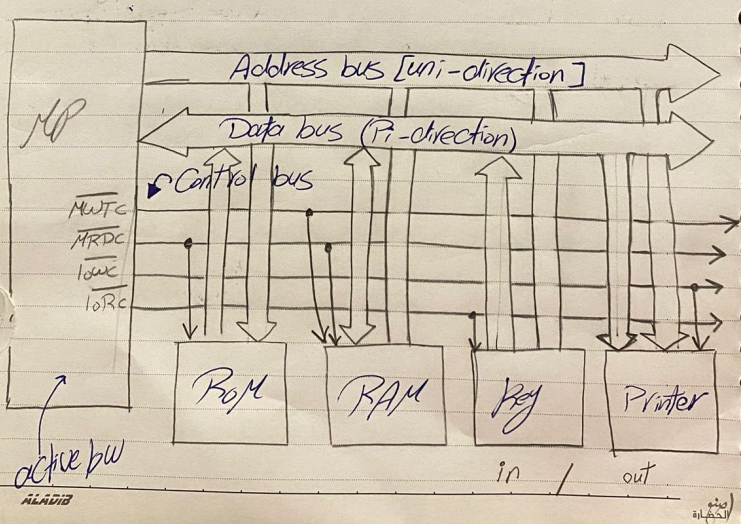
* Computer system → Processor ✓, memory ✓, I/o

I/o → Connect to CPU → ex GPIo
General Purpose input output [control input & output Pins]

---

* Bus set → Address bus, data bus, control bus





Arch

الأسرع

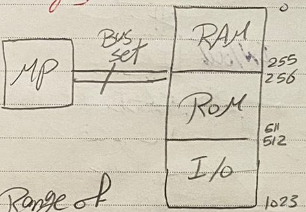
Von-neumen	Harvard

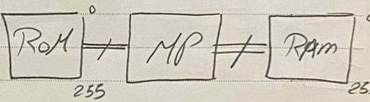
*one memory system

Range كل واحد ليه خاص بيه "memory map"

"Port map"

*





Range of address * كل جهاز ليه أقدر أوصله منه خاص ليه

*معرفش أكلم أكتر من جهاز في نفس الوقت لما أخلص خالص fetch & decode & excute أقدر انتقل للـ address التاني

*أقدر أكلم الاتنين سوا عادي

*المشكلة هي:

ازاي اعرف أنا رايح للـ Rom ولا Ram بالـ address

→ by SW الكود

RAM → Assem load / store

RoM → Assem R/W

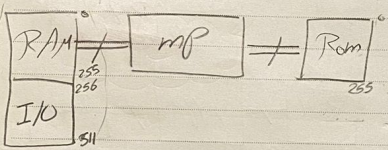
وعلشان أكتب C وهو هيترجم load store

S.Ib الباب

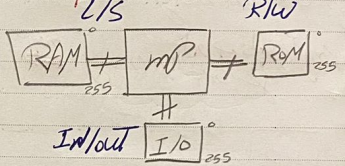
RAM ⟶ C
RoM ⟶ Assem

I/o (Harvard)

memory map "Von-neuman"



Port mapped



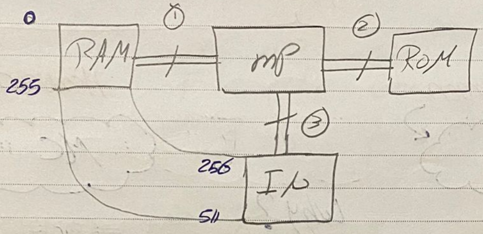
Range of addresses

using C-language

load/store

bus set

* المقصود micro Controller



عشان أتكلم I/O

Bus set ①

C – language (L/S)

دا الي جسم ليك

Bus set ③

Assum (IN/OUT)

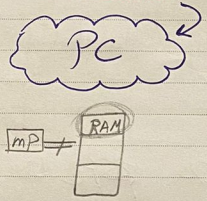
لو عايز بس تفرق انك تتكلم RAM, I/O سوا

"Video 6"

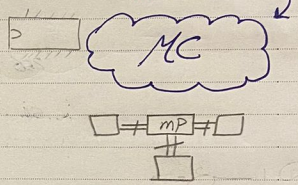
* Using

Arch

Von-neumen



Harvard



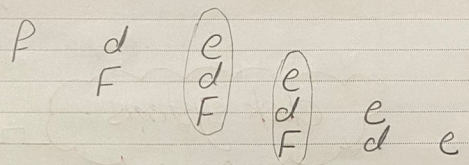
Why?

① بتاخد الـ Code من الـ hard disk وبعدين كده بتتعامل مع الـ RAM يعني مش محتاج إباتع Von-neumen يعني مناسب للـ PC

① الـ harvard سريع جدًا عشان هو بيتعامل مع كل حاجة لوحده

وده مناسب للـ MC

Pipeline → F d e in one cycle



Speed ↑
Performance ↑

So

→ * Von-neumen Can't support Pipeline

* Harvord support Pipeline

* RISC support Pipeline (one cycle)

* CISC Can't support (Complex)

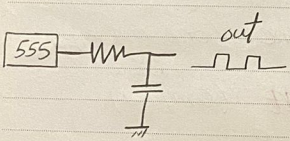
* Instruction → f → d → e → depend on clock [square wave]

clock systems

Electrical

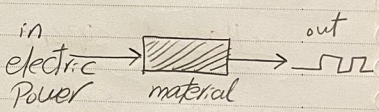
* RC-oscillator

timer 555



~~mechanical~~ mechanical

* material



	Electrical	Mechanical	
Type	RC-oscillator	Ceramic Resonator	Crystal oscillator
Cost	بعاوفرة ممكن ↓ (✓)	in-between	↑ (X)
Accuracy	↓ (X)	in-between	↑ (✓)
setting time	↑ (X)	in-between	سريع ↓ (✓)

ال هيحدد عشان استخدم انهي نوع هو noise imunity
→ Tempreture
→ EMI electric magnetic interferance
→ Vibration

* Resistance, Capacitance → effected by temp & EMI

* Ceramic, Crystal → work with Vibration

noise immunity	RC-oscillator	Ceramic Resonator	Crystal oscillator
Temp	↓	↑ (r)	↑ (r)
EMI	↓	↑ (r)	↑ (r)
Vibration	↑ (r)	↓ (x)	↓ (x)

"Video 7"

Connection

memory mapped — Used at ES

Port mapped

* range of addresses

(base address + offset)

base address :- Connect me to Peripheral

offset :- Connect me to all registers inside Peripheral
each register has different offset

Peripheral :- hardware + Registers

Type of registers :-
* out data register
* in data register
* direction register

→ to any Peripheral

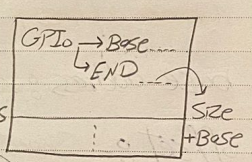
Digital ⟶ give me TRM

TRM :- Technical Reference manual

Reading

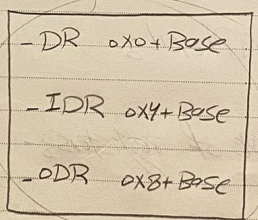
TRM ⟶ Specs

① ↳ memory map (start + end) each peripherals

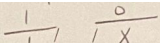


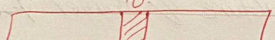
② ↳ Peripheral GPIO

Software ⟶ Register



"Video 8"



*write 1 at  ODR.

1) Using numeric memory address directly.

```
#define GPIo_ODR (Base+offset)
Void main()
{
((Volatile unit32_t*) GPIo_ODR) |= (1<<16);
}
```

2) using d-refrance in #define.

```
#define GPIo_ODR *((Volutile unit32_t*) (Base+offset))
Void main()
{
*GPIo_ODR |= (1<<16);
```

3) using structure & union & pointer.

```
Typedef struct{
        Volutile unit32_t ODR;
        } GPIo_Typedef;

#define GPIo1 ((GPIo_Typedef*) (Base+offset))
Void main(){
        GPIo1->ODR |= (1<<16);
```

[4] Using Structure & Pointer for all Register.

```
#define GPIO_ODR (*(volatile unsigned int *)(Base+offset))
Void main()
{
*GPIO_ODR |= (1<<16);
}
```

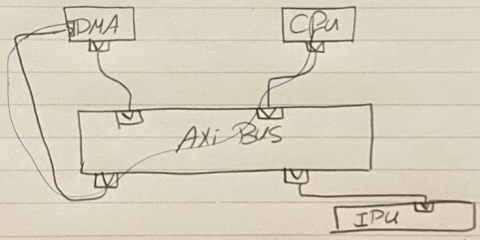
"Hardware Port"

Two Type
① master
② Slave

Protcal Type
* Axi
* AHB
* APB

master :- transshiated transaction ⇄ R (Address, data, size)
W

Slave :- sends the data for size



DMA :- it help CPU to Performance Function

AMBA :- "The ARM advanced MicroController Bus Archeture"

→ reusability (↑)

Standard AMBA → connect between diffeRant Buses

diffeRant BUS → BW → fmax → (Block digram)
→ latency →

ALAAIB