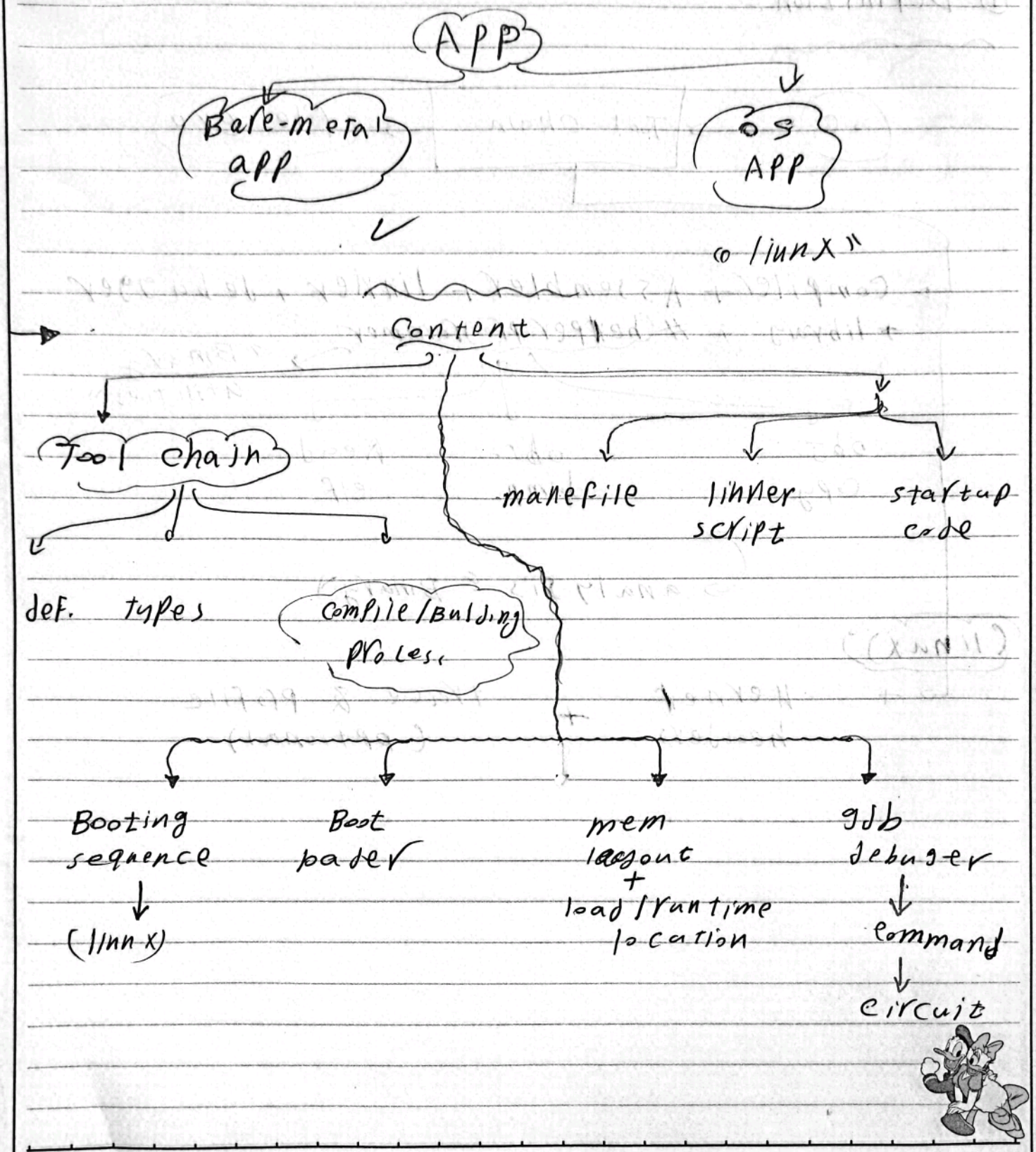
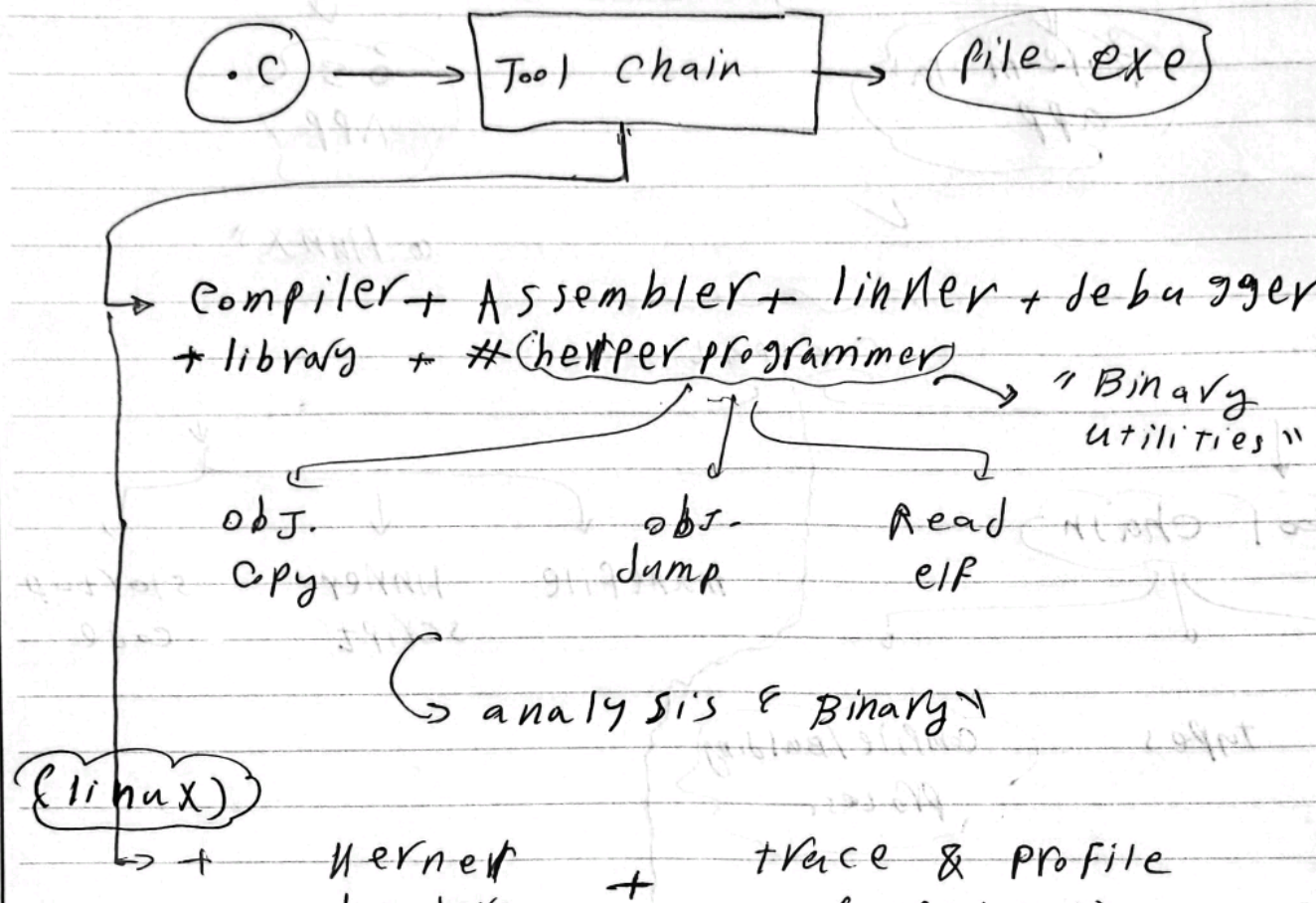
Embedded C

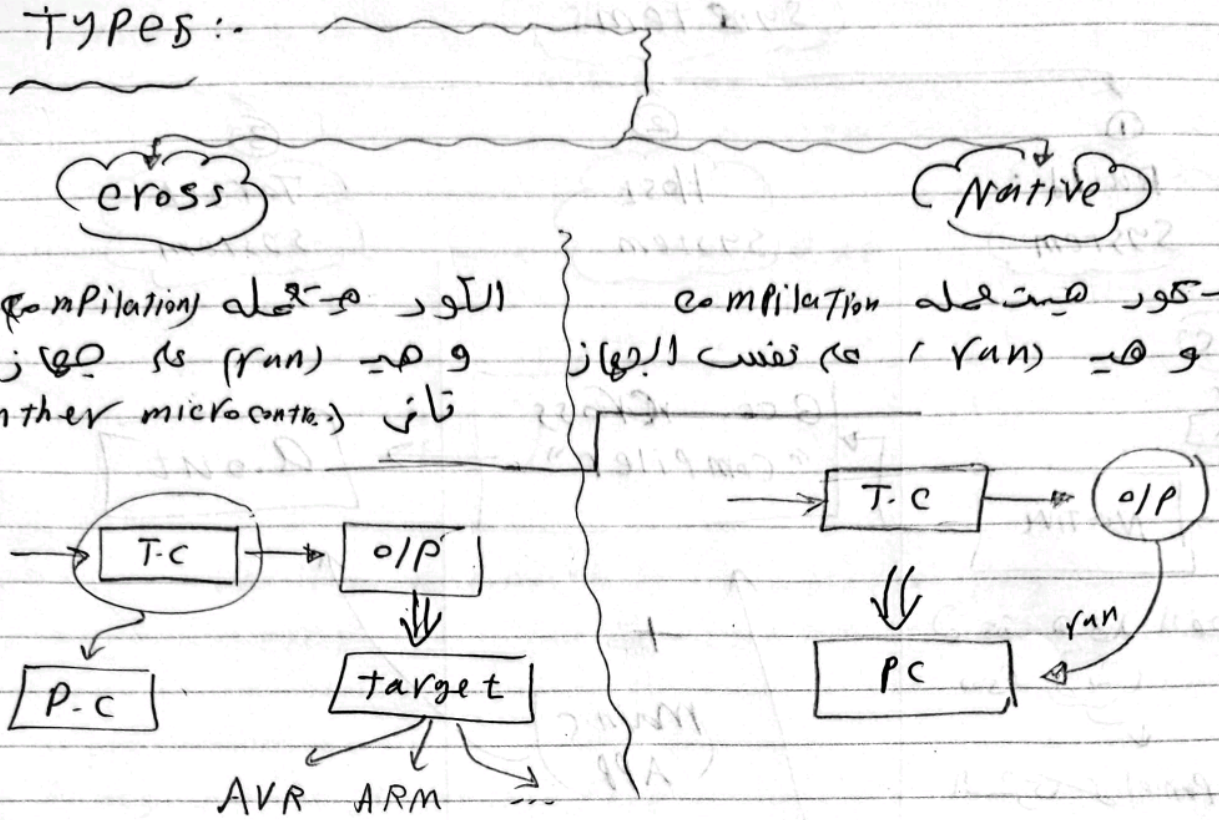

APP
Bare-metal app
OS APP
"linux"
Content
Tool chain
def. types
compile/Building Proles.
manefile
linker script
startup code
Booting sequence
(linux)
Boot pader
mem layout + load/runtime location
gdb debuger
command
circuit

"Tool chain"

① Definition :-



(b) Types :-


cross
Native
الكود هيتعمله compilation
و هيه run على نفس الجهاز
الكود هيتعمله (compilation)
و هيه (run) على جهاز
تاني (on ther microcontr.)
T.C
o/p
P.C
target
AVR ARM ...
T.C
o/p
run
PC

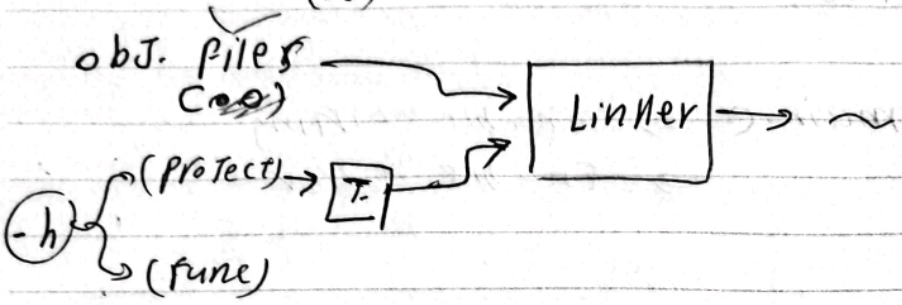
libraries

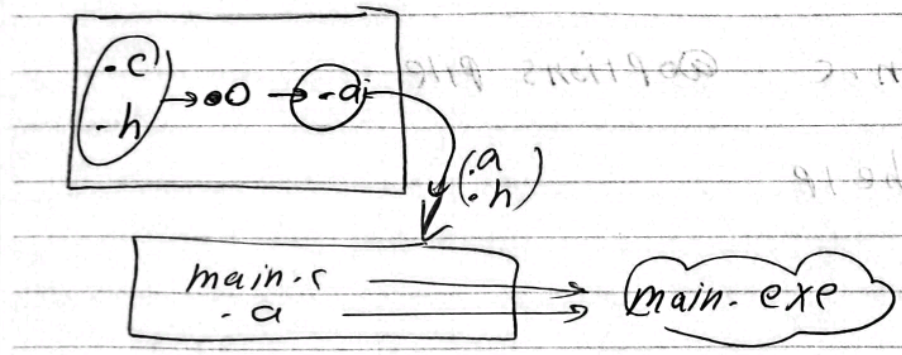
static (.c) ✗✗

Dynamic (.c) ✓✓

(static lib)

(.a) (.o)






```
> gcc -c CAN.c -o CAN.o
```
لـ إنشاء الـ (object file)

```
> ar rcs lib_can.a CAN.o
```
لـ ملف الـ (static library)

```
> gcc main.c lib_can.a -o main.exe
```
لـ تنفيذ (run) الملف

* CMD ⟶ gcc

① gcc main.c ⟶ a.exe

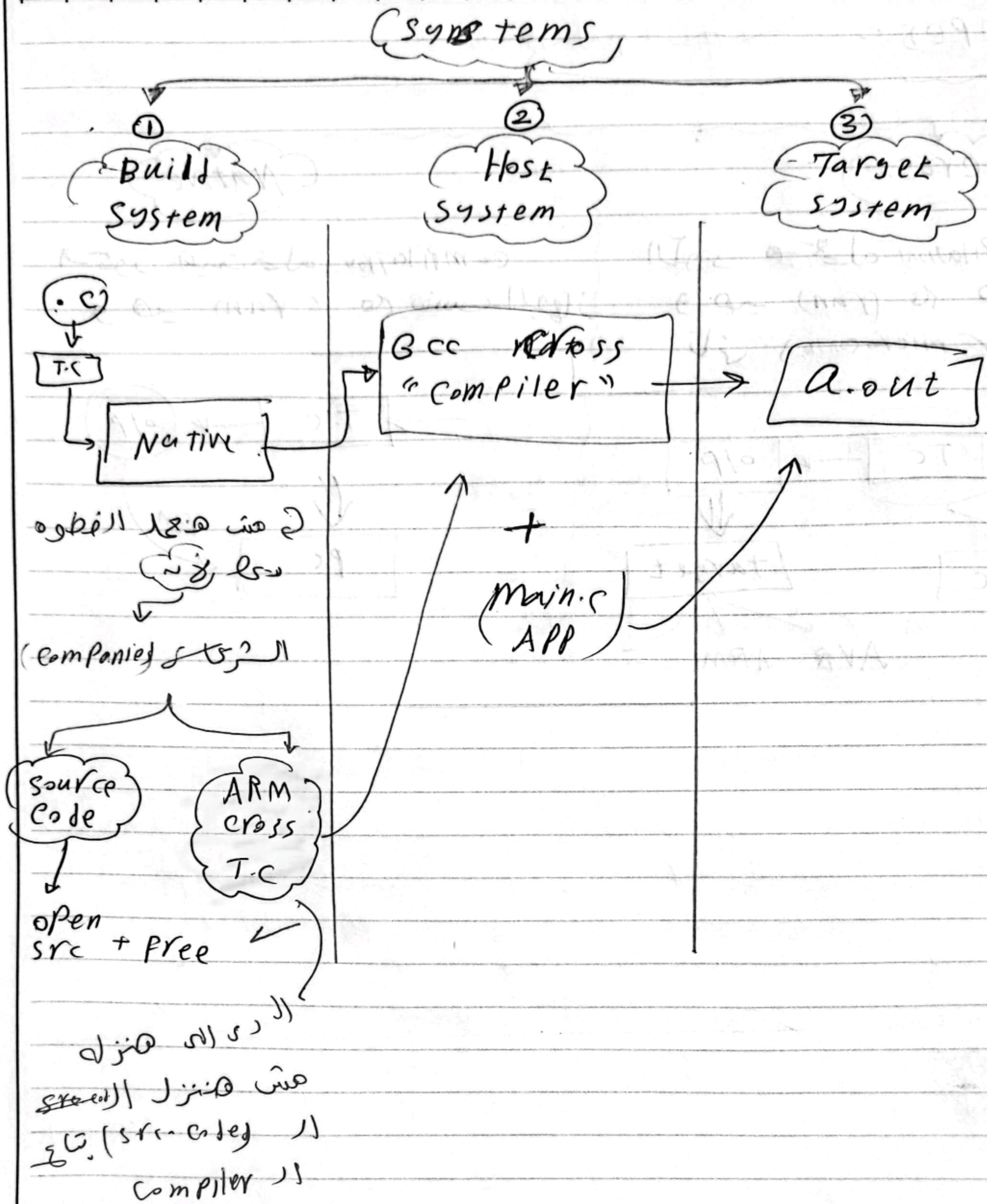
② gcc main.c -o main.exe

③ gcc main.c -I /usr/share/... (header File) ← علشان أعمل include

④ gcc -wall main.c ⇒ Enable warning
// -werror ⇒ Enable error

⑤ Pass options using file
↳ gcc main.c @options file

⇒ gcc --help


Systems
① Build System
② Host System
③ Target System
.c
T.C
Native
Gcc cross "compiler"
a.out
+
Main.c App
لو مش هعمل الخطوه دي لأنه
الشركات (companies)
source code
ARM cross T.C
open src + free
الي هنزله
مش هنزل ال
(src-code) ال بتاع
ال compiler

* Processor Directives

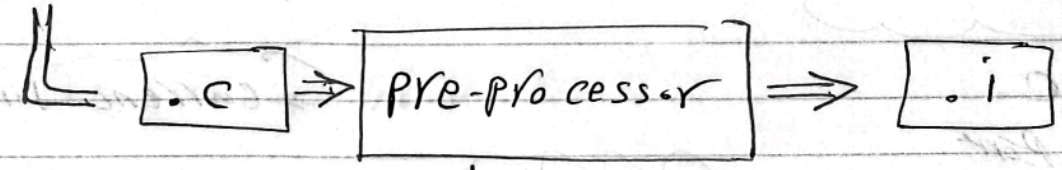
1) → #include

2) → # define

3) → Conditional directive → #if #elif #else
#ifdef #ifndef
...

4) → #error
#warning

5) → string → #
Concatenate → ##



↳ Text replacement ≡ #

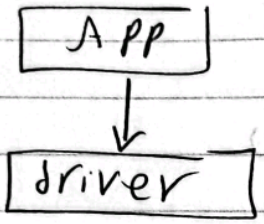
سلسلة

#pragma ⇒ compiler directive

ARM Tool Chain

arm-none-eabi-gcc

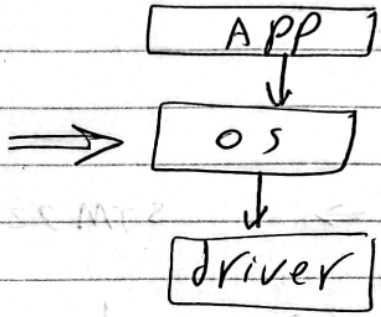
(Bare metal APP)



arm-linux-gnueabi

(+)

(OS APP)



موضوع الدرس : اليـــوم : التاريخ : / /

Hint

⇒ ARM Cross Tool Chain

GNU Gcc
↳ free/open source
commonly used

armcc
↳ Kiel (license)

→ IDE ⇒ STM32 cube (V)

→ without IDE ⇒ download (V)

2] #define (macro)

a) object like macro:

```
#define X 10
#define size 256
```

b) function like macro

notes

```
#define X 10
```

① `int y = X;` ⟶ `int y = 10;`

`int X = 5;` ⟶ `int 10 = 50;`

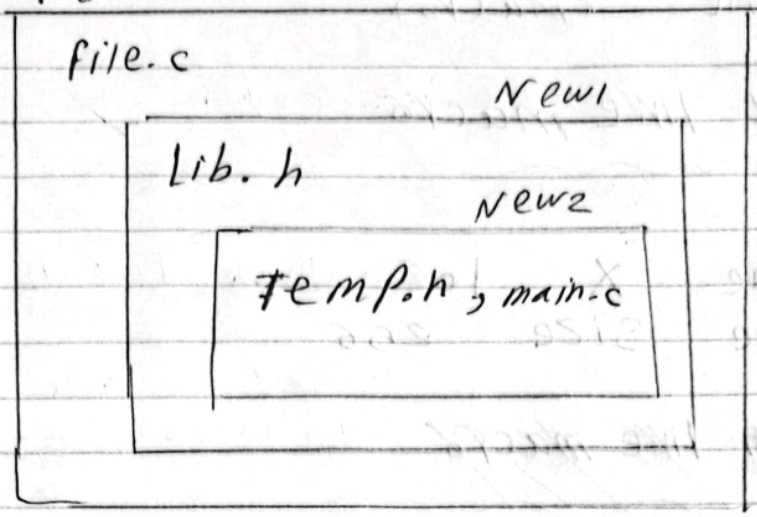
`int Xy = 10;` ⟶ No replacement

`printf("X =");` → // //

`int X = 8;` → // //

↳ capital

P:



@ → file.c → temP.h

⇒ #include "new1/new2/temP.h"

@main.c → lib.h

#include "../lib.h"

↳ back

1] #include

↳ To include header file (.h)

#include<file.h>

(preprocessor) ↳ ال

هيدور في ال standard Library

↳ Built-in Files

#include "file.h"

هيدور في ال current directory

لو ملقاش هيدور في ال (std. lib)

#include "Path"

"Path"

Absolute Path

↳ PC

Relative Path

↳ current directory

موضوع الدرس : اليـــوم : التاريخ : / /

Typedef	macro (#define)
By compiler	pre-processor
→ limited For give symbolic names → types	used for values as well & types
typedef struct std* ptr;	#define ptr struct std *
ptr* m, n; ↓ ↓ // Two pointers //	ptr x, y; struct std *x, y ↓ Pointer Variable of struct std type

* Comparison :-

# define	enum
Text replacement @ pre-processor	Text replacement @ compiler
We can use → float ↳ sentences ↳ ...	must be const int. (Not float) error
No memory space	→ size = int size
X	ممكن استخدمها inside switch (نعم)
X قيمة واحدة ← (خطأ واحد) ← ↳ خطأ واحد	same value different names
any value	$[-2^{n-1} : 2^{n-1} - 1]$

* #define Z 10
  #define X Z
  → (order doesn't matter)

int y = X ; → int y = 10

notes

- #define X 10
  #define X 20
  → // warning //
  لـ هيا خد أخر قيمة

#undef X ← تستخدم ← لتجنب المشكلة دي (warning)

↳ remove the define of the macro

(...) ← ANY_thing else

(Var) هي تخزينها في ال (Value)

(__VA_ARGS__)

#define marem (...) printf(__VA_ARGS__)

```
void main()
{
    marem ("hellooo\n");
}
```

ex:

```
#define fun(a,...) printf(__VA_ARGS__,a)

fun(x, "x=%d");
```

↳ printf("x=%d", x);

Note ⇒ " Concatination operator " ⇒ "\"

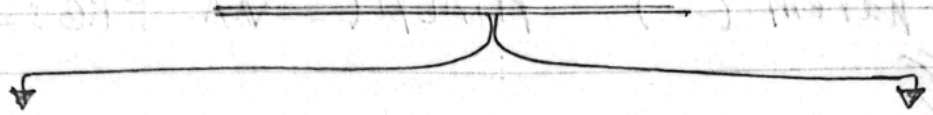
Back slash

#define ___ (-,-) ___\

___\

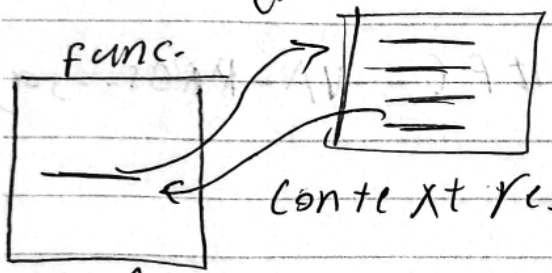
___\

___



المميزات .MACRO

1) Fast in exec.



time (↑)

العيوب dis-MACRO

1) code size (↑)

(.TEXT)

2) No type change

3) Deffifilut to debug

طيب امتى استخدمها

لو ال (code size) قليل

وعامل ال (time) مهم

(2) function like macros:

↳ "parameterized macro"

```
#define Add(x,y)    x+y
```
↳ No space

```
int z = Add(x,y);  ⇒  z = x+y
```

hint: `#define Add__(x,y) x+y`

```
int z = Add (7,4)
int z = (x,y) x+y (7,4)
```
X

3] Conditional directives

① #if cond-1
#elif cond-2
#else
#endif

→ pre-processor

notes

① Comment:-
* We can't make a comment using it :-

#if 0
#endif

2) Configuration → drivers

if	#if
Compile	pre-processor
check val	check macros Not values

ex: #define X 10
#if X > 70

3] Conditional directives

① #if cond-1

#elif cond-2

#else

#endif

→ pre-processor

notes

1) Comment:-

* We can't make a comment using it :-

#if 0

#endif

2) Configuration → drivers

if	#if
Compile	pre-processor
Check val	check macros Not values

ex: #define # 10

#if X > 70

⇒ Pre-defined macros

↳ Built-in ⇒ --DATE--
--FUNC--
--FILE--
--LINE--

التاريخ : / / اليـــوم :

5) stringification & concatination ?

\# ##

① string :-

```
#define printf(x)   printf(#x)
printf(Ali)  ⟶  "Ali"
```

② concatination

```
#define conc(x,y)   x##y
int x = conc(3,8);
```

int x = 38;

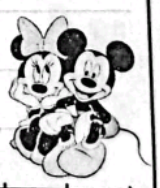
Port driver

4) #error → stop compilation + error message
#warning → warning message

↳ used with → #if
↳ Not (if)

#if (⟶
#error " ⟶ "
✓

if (⟶
#error " ⟶
✗
هيتنفذ كدا لو (false)



موضوع الدرس : اليـــوم : التاريخ : / /

② #ifdef
#ifndef

↳ used as (header file guard)

#ifndef FILEMARCRO
#define " "

#endif

#pragma

↳ Compiler directive

① #pragma optimize ("", off)

↳ "No optimization"

② #pragma once ⇒ replace file guard

↳ يمكنك استخدامها كـ File guard ولكنه ليس الأفضل (Compiler dependent) لأنها تختلف عن (#ifndef)

③ #pragma startup [ ]
#pragma exit [ ]

startup

64 > 65

exit

64 < 65

#pragma start

④ Add memory_section

Usage of Preprocessor Directives :-

① Configuration :-

```
#define size 256
char arr[size];
```

② Readability → (Professional code)

③ Portability

↳ typedef unsigned char u8;
↳ #define u8 unsigned char

hint

#line → يحيد قيمة السطر الحالي

↳ ينفع في عملية الـ debugging

hint (common mistakes)

```c
#include <stdio.h>
#defin max(a,b)   a>b? a:b

int main()
  {
      int x= 9,   y=7 , z;
      z= max(x,y) * 2;
      print("z=%d", z);
  }
```

exp. value ⇒ z = 18

but actually ⇒ z = 9

لأن ال z هتكون قيمة ال (max) أولاً مش تضرب قيمة ال max * 2

to avoid ⇒ case ( )

⇒ #define max(a,b)   ((a)>(b)? (a): (b))

(z = 18)

② Middle Stage :-

1) Symantic Analysis :-

↳ check → logical struct (datatypes)
→ IR

مثال
هيدي تحذير (warning) لو عرفت Ptr بـ char وشاورت بيه على (int)

2) Optimization :-

↳ why? → code-size (↓)
exe.-time (↓)
memory size (↓)

↳ How? → "multi-level process"

مثال :
1) remove dead code ((var زي return بعد))

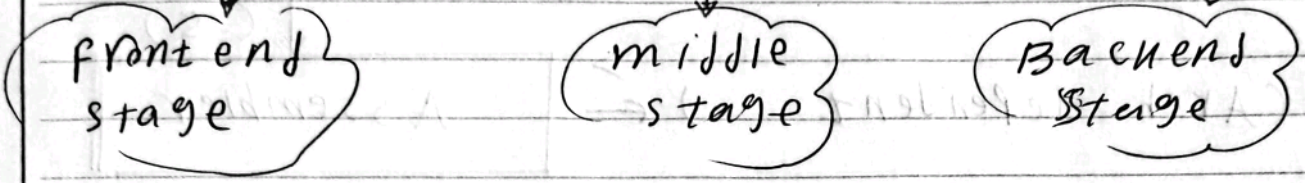
2) inline expansion of func. (exe. هتسرع شوية الـ) (ولكنه الـ code هيكبر)

3) Register Alloc (GPR) → (General Purpose Reg.)

4) loop unRolling → هتحول الـ ForLoop لـ
له مجموعة من الـ
لـ exe. يسرع الـ ، ولكن حجم الكود هيزيد

* preprocessor
  - ↳ Text replacement (#)
  - ↳ concatenation operator (\)

* compiler



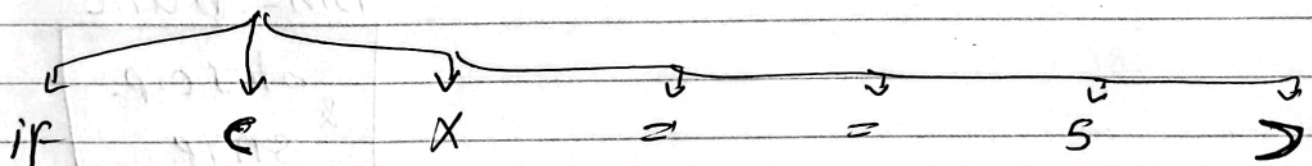
① Frontend stage :-

↳ 1) source code Parsing

o/p → [Parse Tree] → (symatic Table) → IR Program

ex:

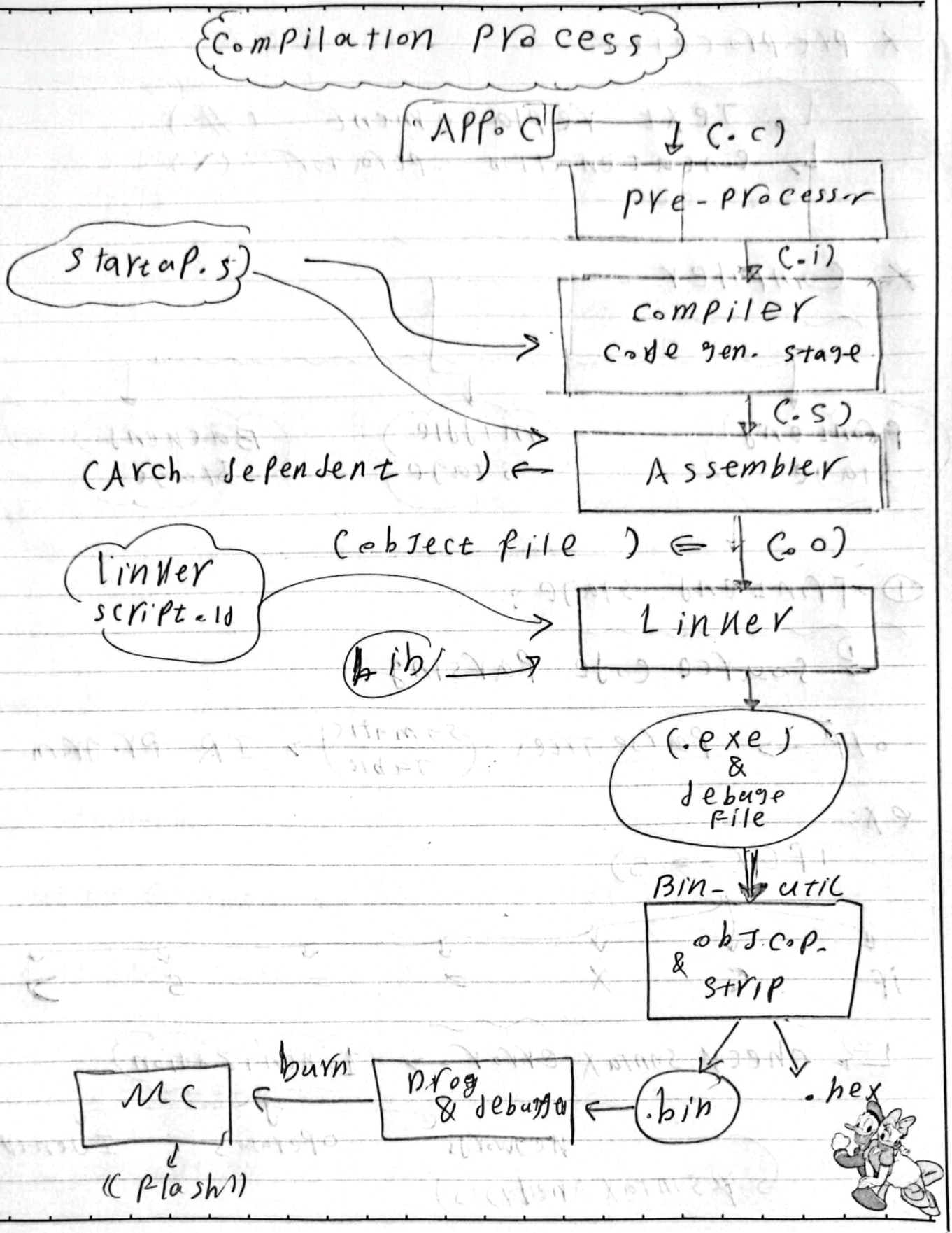
if (x == 5)

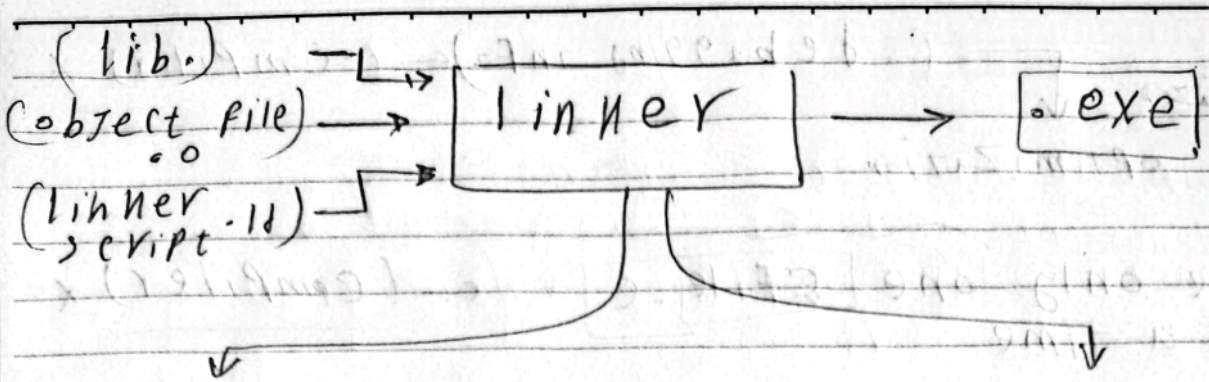


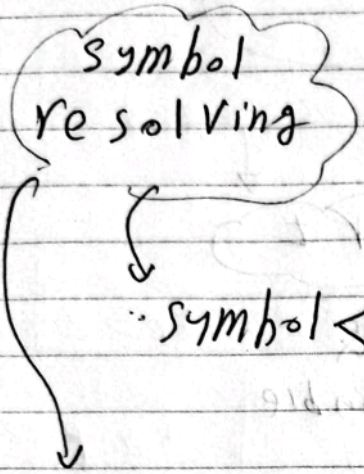
↳ 2) check syntax error → Tokenization

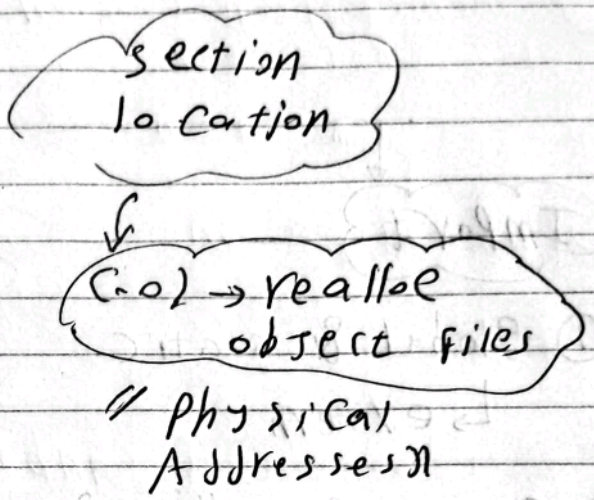
Keywords, operators, Identifier

↳ (syntax Analysis)


Compilation Process
APP.C
(.c)
Pre-Processor
(.i)
Startup.s
compiler
code gen. stage
(.s)
(Arch dependent)
Assembler
(object file)
(.o)
linker
script.ld
lib
Linker
(.exe)
&
debuge
file
Bin-util
objcop
&
strip
.bin
.hex
prog
& debugger
burn
MC
(Flash)







Compilation flags: (gcc --help)

- -E → preprocessing only ⇒ (.i)
- -S → compilation only ⇒ (.s)
- -c → compilation + ASS ⇒ (.o)

* (compiler) ← (debugging info)
↓ محتاجها
optimization

* (compiler) ← parse only one .c file at a time

Import

① global & static
↳ extern

② func. (calling)

⇓

(معلومات من فايلات تانية)

Export

① global & static variable

② func. (implementation)

③ debuging info;

⇓

(معلومات من الفايل (file))

③ Backend stage

① Code generation

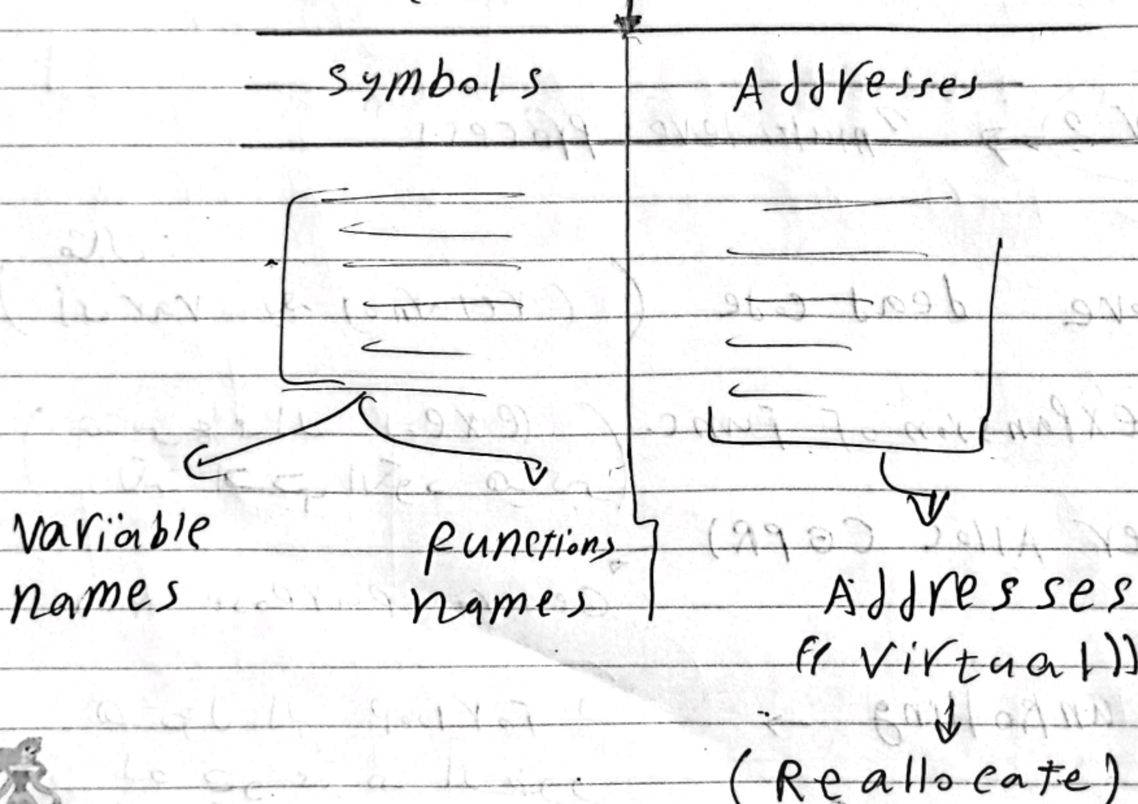
→ convert (code.c) ⟶ Assembly

② memory allocation

└→ mem. segments → .data
                                  → .bss
                                  → ....
                                  ....

ملاحظات من الـ (compiler)

(symbol tabel) ⟵ (compiler)



(TWO cases)

Case ① Case ②

Case ① :

Developer → Entry Point → (Baremetal sw)

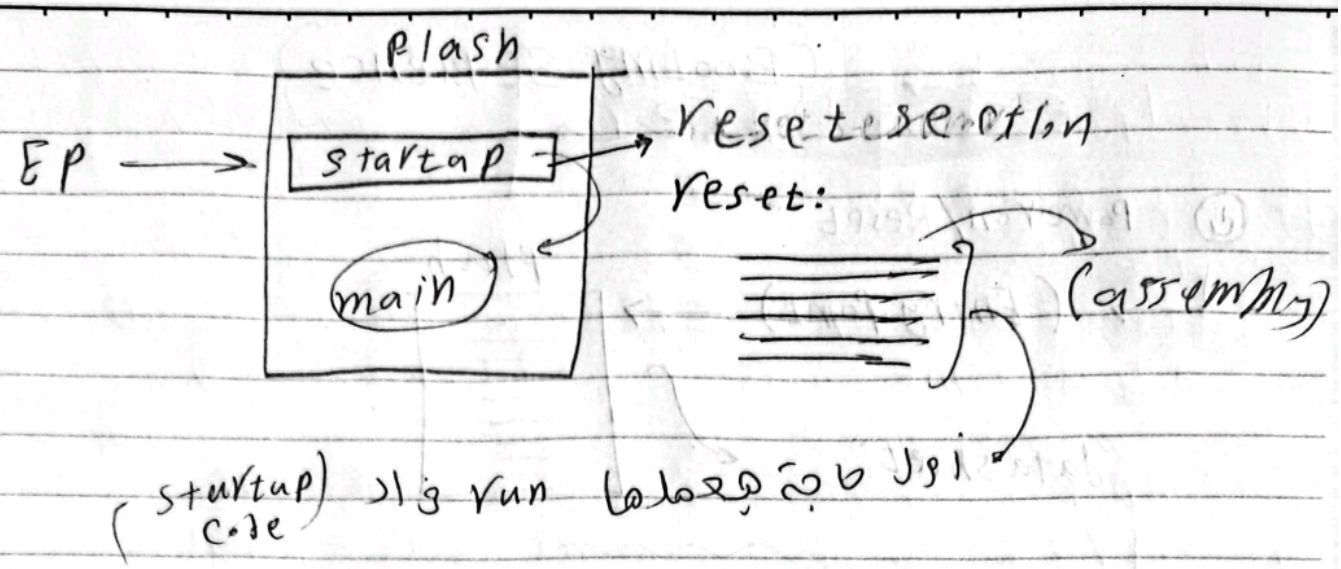
Entry Point → Reset:

(Baremetal sw) ⇓ startup code + APP (main)

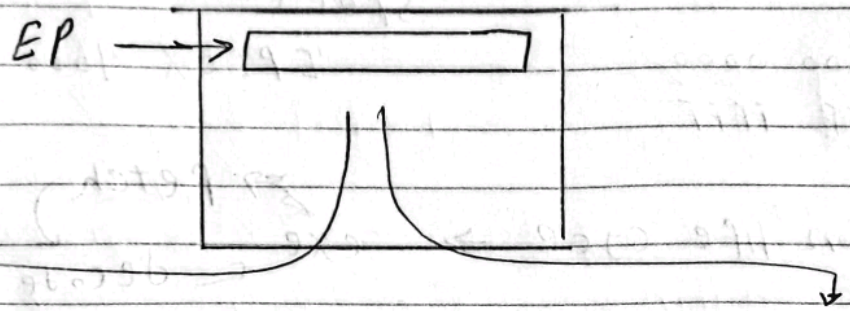
Reset:

1) Initialize stack (SP)
2) Copy (.data) Flash → RAM
3) Reserve (.bss) RAM
4) branch / Jump → main

موضوع الدرس : اليـــوم : التاريخ : / /



(startup code) أول حاجة هتعملها run هو اد



① Bootloader ⇓ (OS) بدون (SW) [startup code + main] ⇐

② your Baremetal SW ⇓ [startup code + APP.]

Booting sequence

① Poweron/Reset

↳ (Entry point) → Flash

"data sheet"

ex:

flash:

0X 0000 0000

0X FFFF FFFF

specs:

EP: 0X 1000 0000

* Istruction life cycle ⇒ fetch → decode → exe.

binary file (.bin) → burn → MC

Flash

EP →

① (startap code) → (.s) ⇒ "if stack Not initialized yet"

↳ (.c) ⇒ "if stack initializes" (ARM cortex)

② code → (.text)

③ global & static (init ≠0) → .data

④ global & static (unintialzed) → .bss

Two Running modes

(Rom mode) (RAM mode)

(Rom mode)

- **RoM**
  - Entry point
  - Exec. code + const. data
- **RAM**
  - Stack + changing data

(RAM mode)

- **RAM**
  - exe. code & stack + changing data
- **RoM**
  - Entry point
  - Boot loader
  - & code image (os/sw)

RoM ال ← موجودة في

SD card, ethernet, flash ← onther location ← RoM ال

(compare)

RoM mode	RAM Mode
- very simple	- complex
- reg. smaller memory	- large code
- Fixed code Address	- Relocatable mode
- Relative small code	- Fast.

Case ②

Developer ⇒ Entry Point → (Boot loader)

startup code → main

1)
2)
3)
4)

Boot loader

Boot

loader

load (source → destination)
(@ runtime)

(Boot loader) ⇐

Initializer ①

Prepherals (modules) ↳ SPI

⇐ loading ②

Flash → load → os/sw → load → RAM

os → linux (embeded)

(startup code) ← Jump ③

↳ sw → main

hint ⇒ reusable memory space