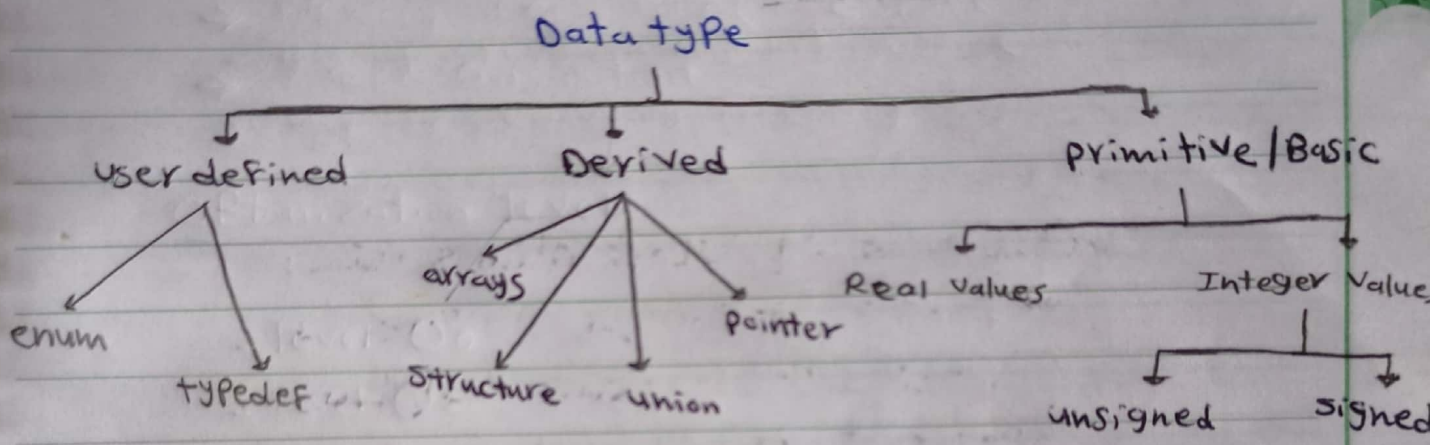
typedef ~~Enum~~ Command :-

typedef keyword in C used to creat an alias-name to user defined data type to primitive and Derived datatype.



Syntax

typedef <primitive or drived datatype> <new_name>

Some Notes to make your Code more readable

1. Make NEW_NAME Capital letter or like that New_Name

2. put (_t) after Name
   ex Name_t

3. Put (S) Capital if structure and (U) if union at beginning → Sname_t
   if it is a Enum put (_e) → Name_e

الموضوع : التاريخ :

Header protection :

Syntax

```
#ifndef T_H_
#define T_H_

_____
_____
_____

#endif
```

Usage

بنستخدمه عشان في حالة إننا كملنا include لـ file أكتر من مرة في ملفات مختلفة وبينا إحنا استخدمنا الملفات دي في ملف واحد فلو إحنا محطهوش هيحصل Text replacement لـ definition الحاجات أكتر من مرة وده هيؤدي إلى إنه يحصل عندنا compilation error فهو هيساعدنا إن الحاجة يتعمل ليها text replacement مرة واحدة بس والكود ميضربش معانا ايرور.

التاريخ :

الموضوع :

Optimization :-

is a series of actions taken by the compiler on your program's code generation process to reduce :-

→ number of instructions (code space optimization)

→ Memory access time (time space optimization)

→ Power Consumtion

Optimization levels :-

- level O0 zero
- level O1 one
- level O2 two
- level O3 three

Compiler make optimization level by default is → level O0

شوية كلام كدا نفهم الحاجات اللي فوق دي

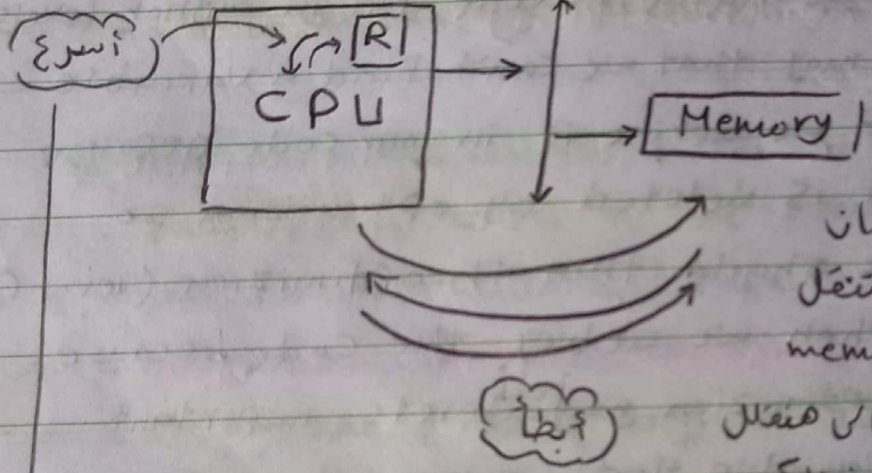
- **reduce number of instructions**

This will reduce .text section that has all assembly instructions so the code size will decrease so the binary File will take less memory in Flash memory and reduce the instructions will make the processor execute the code fast and less the execution time of the program.

- **Memory access time**

if we have a variable and this variable used in number of equations and after every equation we save the result in the memory so this operation will take time so we can save it in a register in CPU to take less time and switch to memory in each operation.

in general purpose register



هنعمل كدا عشان نقلل إننا نفكد ننتقل بين الـ CPU و الـ memory بالشكل ده و بالتالى هنقلل الـ Time بتاع تنفيذ الكود

→ This will reduce power consumption

← الكلام ده كله هيؤدى إلى زيادة الـ Performance بتاعت الكود و هيقلل الـ Code size والـ Memory المستهلكة

الموضوع :

التاريخ :

Optimization level O0 :-

- No optimization
- Not recommended for production if you have limited
  - Code
  - Ram space
- Has fastest compilation time
- This is debugging friendly why??
  because in debugging we find all variables that we define in code but in author levels we may that we can't find a variable that we define it in our code because it is deleted by optimizion so we should turn off optimizion (level O0) when we debug the code because optimizion may delete important variable that we need it in our code. So the logic of the code may be wrong after this.

- A code which work with O0 may not work with O0+ optimization levels.

How to open the debug on keil ??

بعد ما كتبت الكود بتاعنا كل stm32 وعملت build طلع عندنا جوا الفولدر بتاع ال project ملف debug ملف elf. هناخده كوبي ونروح لأي project فاضي على keil وبعد كدا منفتح كل options for target وبعدين بعد كدا نروح على ال output نحط الملف اللي أخدناه كوبي بس هنختلي axf. وبعد كدا نأخد اسم الملف نحطه في اسم ال executable وبعدين نقدر نشغل ال debug

المصرية الإيطالية

debug وبس كدا ال

Optimization level O+ :-

optimization leve O1 :-

Moderate optimization to reduce

- Memory access
- Code space

optimization level O2 :-

- Full optimization.
- slow compilation time.
- Not debugging friendly.

optimization level O3 :-

- Full optimization of O2 + more aggressive steps will be take by the compiler.
- slowest compilation time.
- May cause bugs in the programy.
- Not debugging friendly.

note

لو حبينا عملنا debug وإحنا مشغلين ال optimization بـ level O2 ساعتها ممكن :-

متلاقيش symbols لحاجات إحنا مكوننها وهتلاقي pure assembly وإحنا مش هنفهم حاجة أبدًا.

الموضوع :

التاريخ :

Volatile Type Qualifier

- Volatile keyword is a qualifier that is applied to a variable when it is declared

- it tells the compiler that the value of the variable may change at any time. "without any action being taken by the code"
  So it can't optimize.

- "Always Read From Memory" No optimization possible

Syntax

- Volatile int x;
- int Volatile x;

لو عندنا Variable معين وقيمته مش بتتغير باستخدام الـ code خالص المفروض إنه هيحذفه فاحنا عايزين الـ Variable ده ميتحذفش لأن قيمته بتتغير بس مش بالكود بس بتتغير عن طريق أي حاجة موجودة في الـ Hardware وبالتالي هنستخدم Volatile عشان الـ Variable ده ميتحذفش.

external input

Proper use of C's volatile keyword

- Memory-mapped peripheral registers

لو عندنا module ما فيه بعض الـ registers وفي register عايزين نقرأ منه values فاحنا هنعمل pointer للـ register ده بس ممكن قيمة الـ pointer تتغير في أي وقت فاحنا عايزين نقرأ التعديل ده و نعرف بيه فاحنا هنخلي الـ pointer ده Volatile عشان الـ pointer يفضل موجود وميتحذفش بسبب الـ optimization وكمان نقرأ أي تعديل يحصل في قيمة الـ pointer.

- Global variables modified by an interrupt service routine (ISR)
- Global variables accessed by multiple tasks within a multi-threaded application

---

note

```
Volatile uint8_t *pReg;
uint8_t Volatile *pReg;
```

Pointer to volatile data

Volatile unsigned 8-bit integer

```
int *Volatile p;
```

Volatile pointer to non volatile data

```
int Volatile *Volatile p;
```

Volatile pointer to volatile data

---

note

أى register هنكتب عليه أو هنقرأ منه نحرفه ك pointer to volatile data عشان ميتحذفش باستخدام الـ optimization وعشان كمان للكتير لو يحصل فيه مثال external i/p يسمع في البرنامج ويكون شغال صح.

How To access the register by absolute address ??

ex

write oxFFFFFFFF on SIU register which have absolute address ox3061000

Solution :

1st method :

```
Volatile int *p;
p = (Volatile int *) ox3061000;
*p = oxFFFFFFFF;
```

2nd method :

```
*((Volatile int *)(ox3061000)) = oxFFFFFFFF;
```

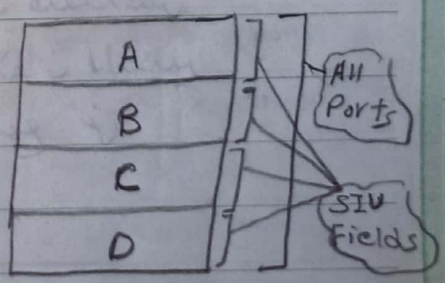
3rd method :

```
#define SIU_Register *((Volatile int *)(ox3061000))
SIU_Register = oxFFFFFFFF;
```

2nd method = 3rd method

4th method :

في الطريقة دى إحنا هنستخدم union وجواه struct بحيث إننا لو عايزين نستخدم ال Ports ع بكمن (32 bit) عادى ولو عايزين نستخدم كل Port لوحده فبرضو عادى (8 bit).



```
typedef union {
    vuint32_t   All_Ports;
    struct {
        vuint32_t   PortA:8;
        vuint32_t   PortB:8;
        vuint32_t   PortC:8;
        vuint32_t   PortD:8;
    } SIU_Fields;
} SIU_R;

Volatile SIU_R * Ports = (Volatile SIU_R*) 0x3061000;
```

- `Ports -> All_Ports = 0xFFFFFFFF;`

هنا إحنا هنكتب كل الـ Ports مع بعض ↵

- `Ports -> SIU_Fields.PortA = 0xFF;`

هنا إحنا هنكتب كل PortA فقط ↵

Usage of Const and Volatile together

```
uint32_t volatile * const Preg1 = (uint32_t *) 0xFFFF0000;
```

Preg1 is a Constant pointer to Volatile data of type uint32_t.

Const is used to guard the pointer From unexpected changes From the programmer.

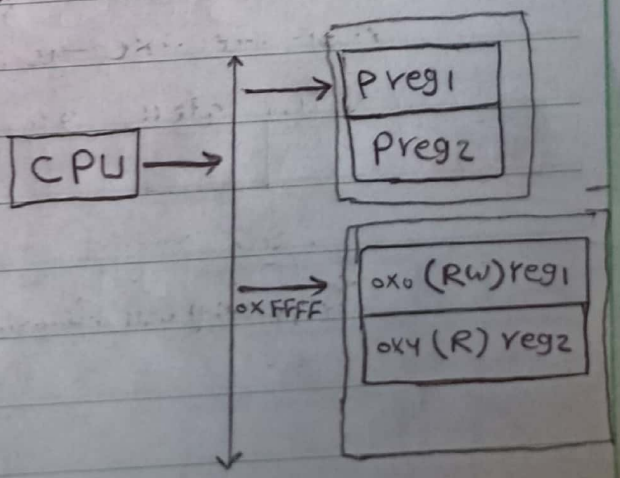
الـ Const هنا هيحمي الـ Pointer من إن الـ user يحاول يغلط ويشاور على مكان تاني جوا الـ memory يعني لو الـ user عمل

Preg1 ++i → This will make compilation error.

Volatile used to tell the Compiler that data pointed could receive unexpected changes So the Compiler has not to apply any optmization on the data.

يبقى الخلاصة إن الـ reg1 نقدر نكتب عليه باستخدام الـ Pointer. ونقدر نقرأ منه عادي باستخدام external I/P

reg1 → (RW) ✓✓



uint32_t const volatile * const Preg2 = (uint32_t) 0xFFFF0004;

Preg2 is a constant pointer pointing to volatile constant data of type uint32_t.

What that means ?? :)

This qualifier (const + volatile) on data.

it is Fixed :)

it can be changed at any time :(

Const means that the programmer can't change the data by the code and if he try to do that the compiler will give (error !!)

Volatile means that the data can be changed by the hard ware "external input" not from the programmer.

طب ايه الـ register اللي يبقى مفيد معاه الوارنيه ؟؟ :)

it is useful for (R (read only access))
register like the status register which is changed from Hardware not from the programmer

يبقى مفيد لو عشان مثلا الـ status register عايزين
الـ data بتاعته تتغير على حسب العوامل الخارجية مش
إحنا اللي نغير قيمته بإيدنا ولو حاولنا نغير قيمته
باستخدام الكود فده يعمل ايرور عندنا

الموضوع : التاريخ :

Tool chain

consists of → compile , assembler , linker , debugger , libraries ,

number of helper programes ← [Binary utilites]

ex:
- objcopy
- objdump
- readelf

the host PC (Ubuntu, redhat, ---) by default is native toolchain.

Toolchain

Cross

in this tool chain we setup it in our PC and we write code on it and it will run on pc but generates code for anther archtecture.

يبقى هى ال Tool اللى بتسطب على code وبتكتب عليها ل archtecture bin File وبطلع ال run عشان يشتغل على anthor arch عشان

ex: Atmel studio

Stm32 cube

Native

in this toolchain we setup it in our PC and we write code on it and it will run on PC and generates code for the same PC.

يبقى هى ال Tool اللى بتسطب على code وبتكتب عليها ل archtecture bin File وبطلع ال run عشان يشتغل على same archtecture عشان

ex : mingw

True or False

The executable File gets From cross tool chain it can run on the same tool chain.

(False)

why لأن ال cross toolchain بيتكب عليها code نطلع منه ال executable file عشان يتحرق كل ال anthor architecture وبالتالي هتبقى False.

Host machine ⇓ دى اللى إحنا بنكتب عليها الكود.

target machine ⇓ دى اللى إحنا عايزين الكود يتحط عليها ويشتغل.

Static library

دى بيبقى binary code لملف ما مكتوب فيه بعض ال Functions وإحنا عايزين نستخدم أى حاجة من ال Functions دول بس بدون ما نكون ال implementation بتاعتهم فإحنا هنعمل static lib نحمل بيها الكواريد ده وده بالاء إن اسمها يبدأ بـ (lib) وينتهى بـ (.a)

وال static libraries دول يتحمل ليهم link مع ال object File جوا ال linker عشان يخرج عندنا ال executable File.

مثال عندنا مثلا لما نيجى نستخدم "printf" فاحنا مش عارفين ال implement بتاعها بس إحنا بنعمل include لـ "stdio" عشان يحمل link بين ال static libraries بتاعتها مع ال object File ويخرج عندنا ال executable File بدون أى مشاكل.

note:

- library object files should not contain a main function
- static library names should start with "lib" and end with ".a" ⇒ ex:- libmath.a

important note

ال linker بياخد ال object الخاص بالجزء المستخدم فقط
مش بياخد ال library كاملة على بعضها.
يعني لو بنستخدم "printf" في برنامج فال linker هياخد جزء
ال printf فقط من "stdio.h" ومش هياخد ال File كامل

To creat a static library file

```
ar rcs lib_name.a File1.o File2.o
```

To add anther object File to the library

```
ar r lib_name.a File3.o
```

To remove an objectFile from the library

```
ar d lib_name.a File3.o
```

To View the object files in the libraray

```
ar t lib_name.a
```

Toextract object files From the library

```
ar x lib-name.a
```

note

ايه الفرق بين (2) بدي

- arm-none-eabi
- arm-linux-gnuapi

arm-none-eabi

ده بيستخدم مع الـ Baremetal applications يعني إحنا بنكتب code هيتحمل على arm ← direct على run

arm-linux-gnuapi

ده بيستخدم في إننا نكتب application مش هيشتغل على arm ← direct الموجود على linux ولكن هيكلم الـ microcontroller .

who is used to write Baremetal application on Arm Board ??

a) arm-none-eabi ✓✓

b) arm-linux-gnuapi

خطوات تنفيذ مثال على الـ static lib :-

١. نعمل 3Files يعني ← main.c , lib.c , lib.h

٢. نكتب Func جوا الـ lib.c ونحط الـ prototype بتاعها جوا الـ lib.h

٣. نعمل call لـ Func دي جوا الـ main.c

٤. نطلع الـ main.o من الـ main.c وهكذا مع الـ lib.c

٥. نعمل staticlib جوا الـ object file بتاع الـ lib.c

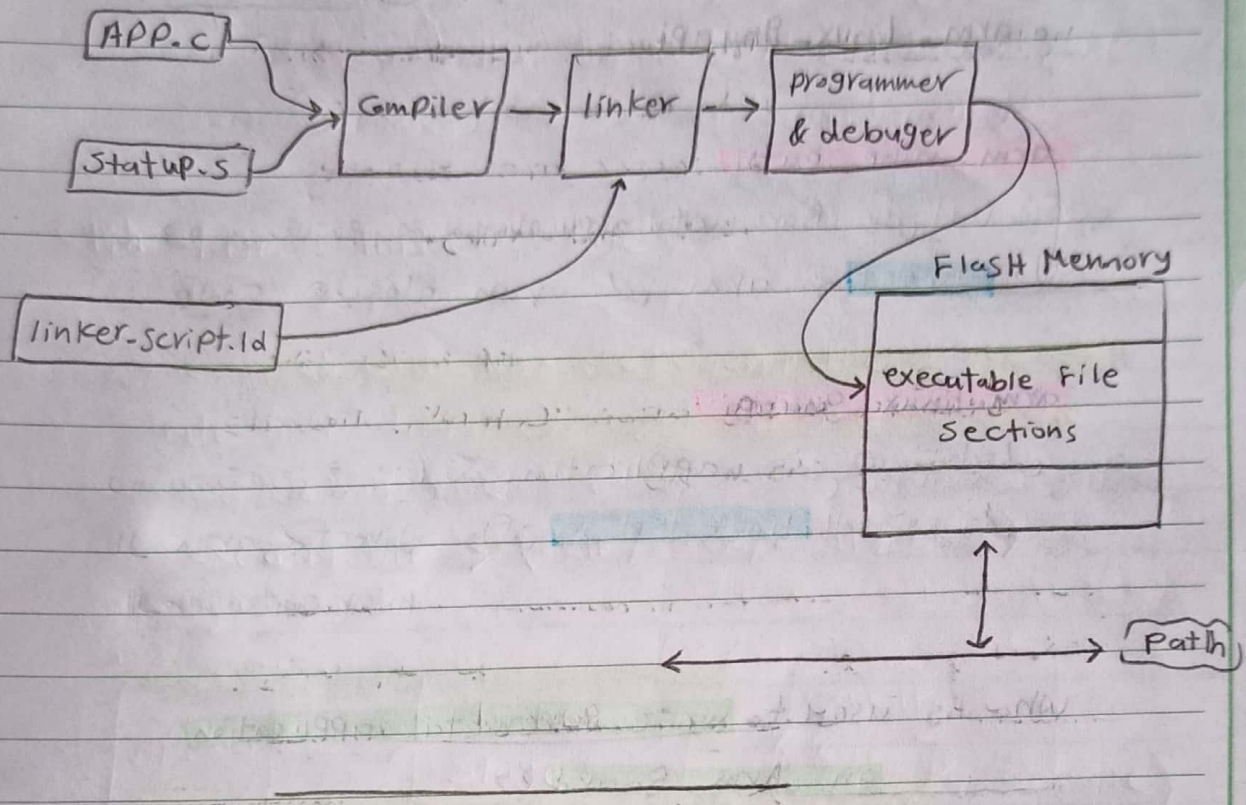
٦. نعمل link بين الـ static lib والـ main.o ونطلع الـ lib.exe

٧. نرن الـ exe. هيشتغل معانا تمام زي الفل.

```
gcc -c -I . main.c -o main.o
ar rcs lib_trystatic.a lib.o
gcc main.o lib_trystatic.a -o lib.exe
lib.exe
```

✓✓

Compilation Process :-



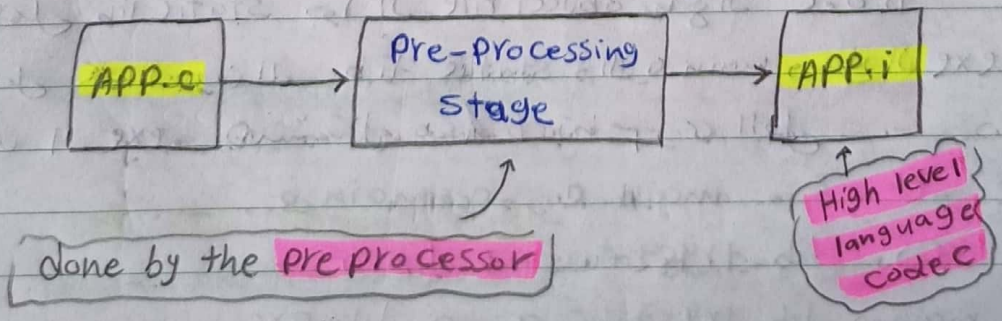
1st Stage Pre-processing Stage

in this Stage ⇒ all Pre Processing directives will be resolved

Macros → #include → #ifdef ,...

intermediate File ← File.i بيخرج عندنا

#pragma lush # بيكون الملف ده خالي من أي

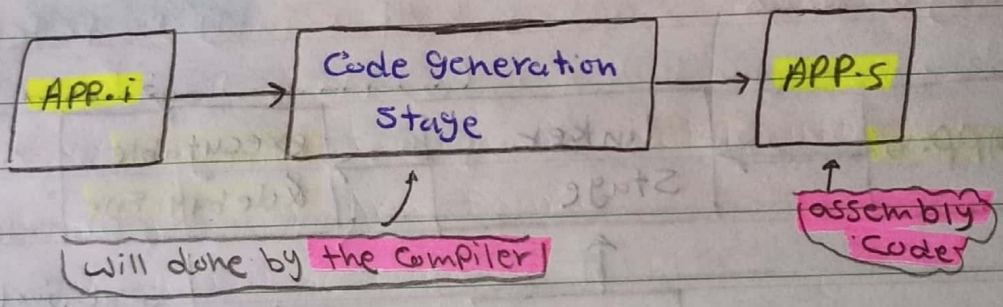


2nd stage Code generation stage

in this stage ⇒ High level language code C will be converted into processor architecture level assembly.

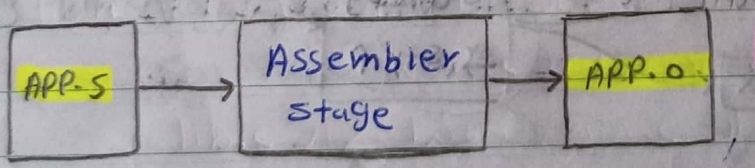
This stage dependent on Architecture

why because every architechture has its own assembly.



3rd stage Assembler Stage

Assembly will be converted into opcode & oprands (binary)



ما معنى الـ Relocatable object File ؟؟

هو عبارة عن الـ object code أو الـ binary code بس بيحتوي على الـ virtual addresses للداتا اللي بتتعمل و مش بيحتوي على الـ physical addresses الـ memory.

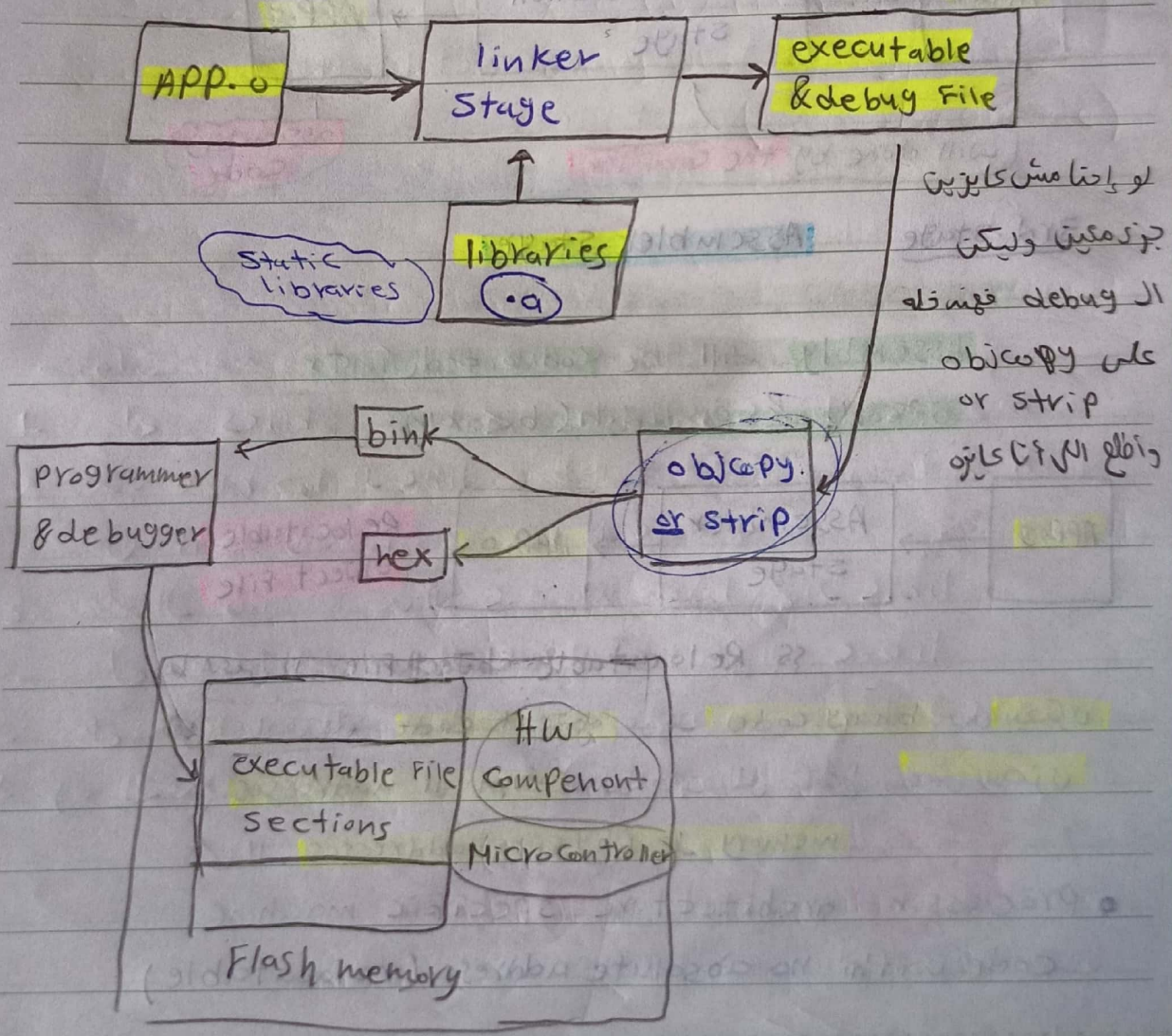
- Processor architecture specific machine code with no absolute address (rolocatable)

المسؤل عن أخذ سكاشن ال assembly من ال object files
ويجمعهم في File واحد ويضبط ال addresses بتاعتهم
مع ال addresses الخاصة بال microcontrollor
هو ال linker script

4th Stage linker Stage

the linker will link between the object file and libraries and out the executable and debug file for us.

executable → (.elf / .axf / .out / .exe)



.map File

it is a file we can get it from after linking stage.

it is include: the actual length of the sections and the names and locations of the public symbols found in the relocatable program.

ex symbols → global variables, Functions, ...

A processor executes what kind of code??

a) assembly code
b) Machine code
c) Eo Code
d) Morse code

---

linker file

- provides details on the memory size and location.
- Maps compiled sections to physical memory.
- can check if the memory was over located.
- can provide the entry point to the program.

---

if a program is linked against a static library then

Machine code of the used function is copied in the executable.
