ARM - 32 bit processors:

32 bit ??

→ Internal registers in register banks are 32-bits

→ Data path are 32 bits

- Data process & operation
- Arithmatic operation like addition, subtraction

→ Bus interface is 32-bit

Bit size:

→ Allows CPU to address memory for an individual process

x-bit can handle 2^x bytes of memory

For ex: 8 bit can hande

$2^8 = 256$ bytes of memory

16 bit

$2^{16} = 65536$ bytes ✓

32-bit

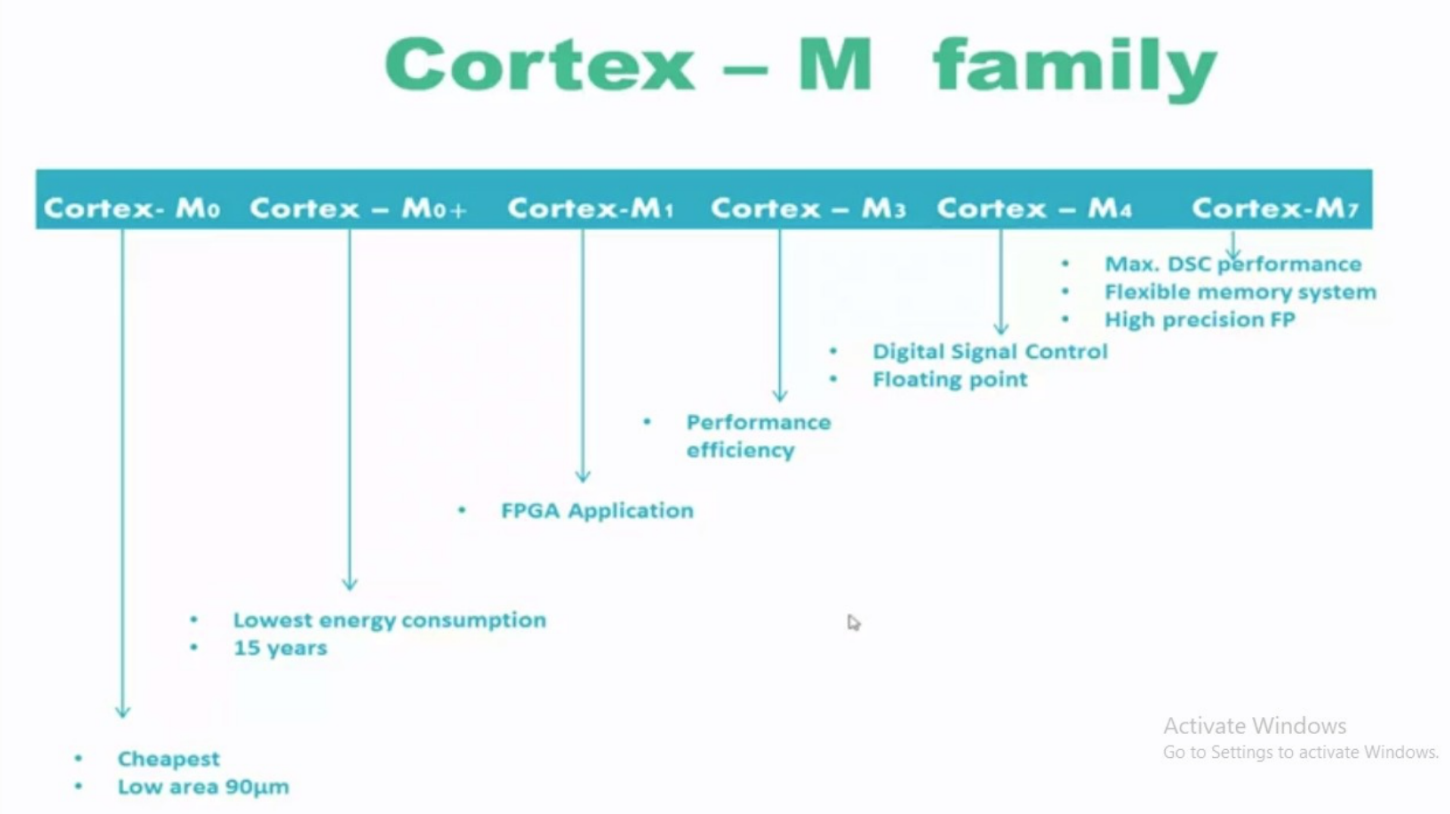
$2^{32} = 4.2GB$

→ so higher the bit size higher the performance

→ So this is very important when we are working with system where performance and response time are key

* CORTEX – M family:

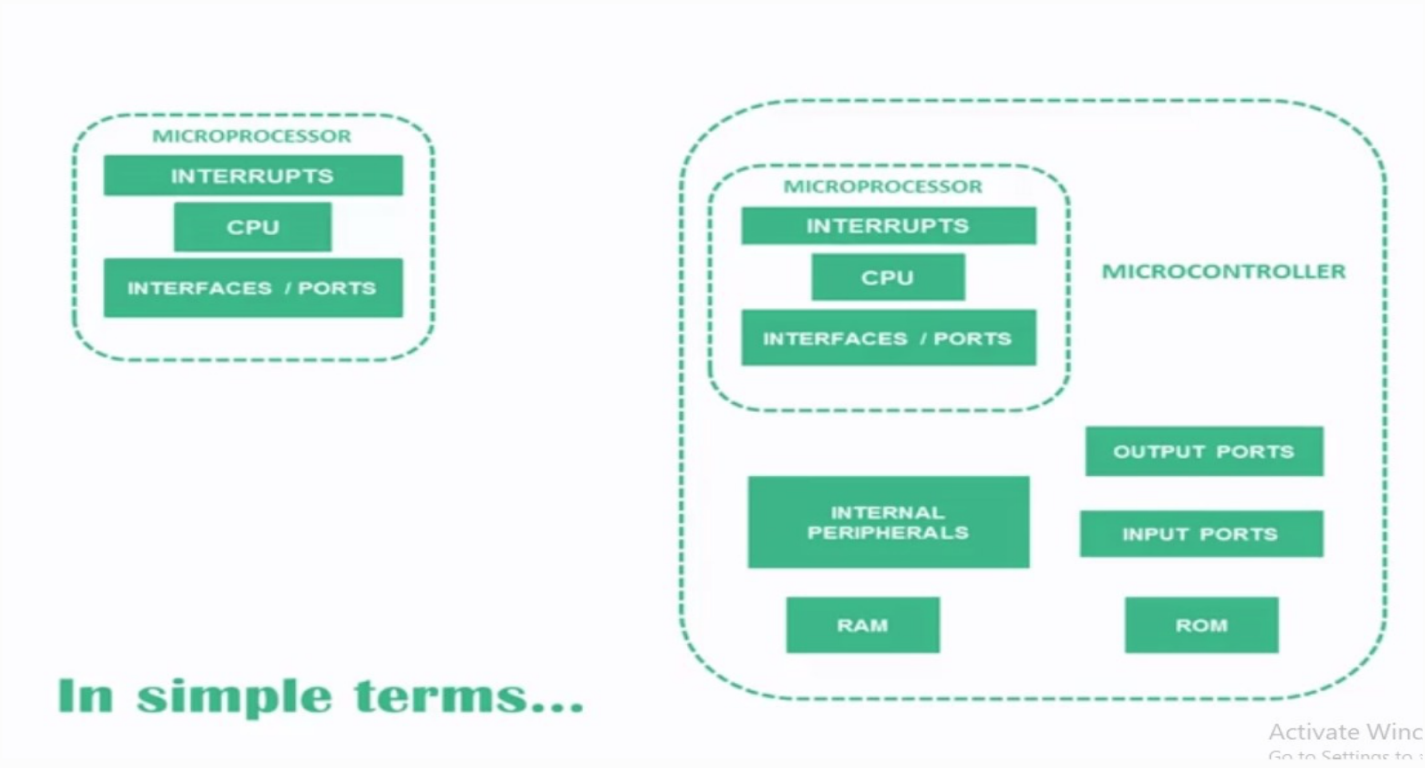
Cortex M0, M0+, M1, M3, M4, M7



Why cortex M1 ??

- High code density (space programme takes from memory)
- balance between energy efficiency & performance
- High interrupts (240) (priority)

* Microprocessor Vs Microcontroller



* CORTEX - M Architecture :

→ cortex - M has separate data and instruction buses hence it is a harvard architecture.

→ Architecture are based on way of accessing memory

i) von_neuman or princeton

ii) Hardcoard architecture

→ Hardcoard architecture allocas simultaneously of both data and instruction.

→ von-neuman has only one line for data & instruction, therefor they have to be scheduled

⊙ Registers:-

→ A register is fast accessible storage inside a processor core

→ data processing & control

→ They are grouped together in an unit called register bank.

⊙ Load-Store Architecture:-

→ when data from memory is to be process

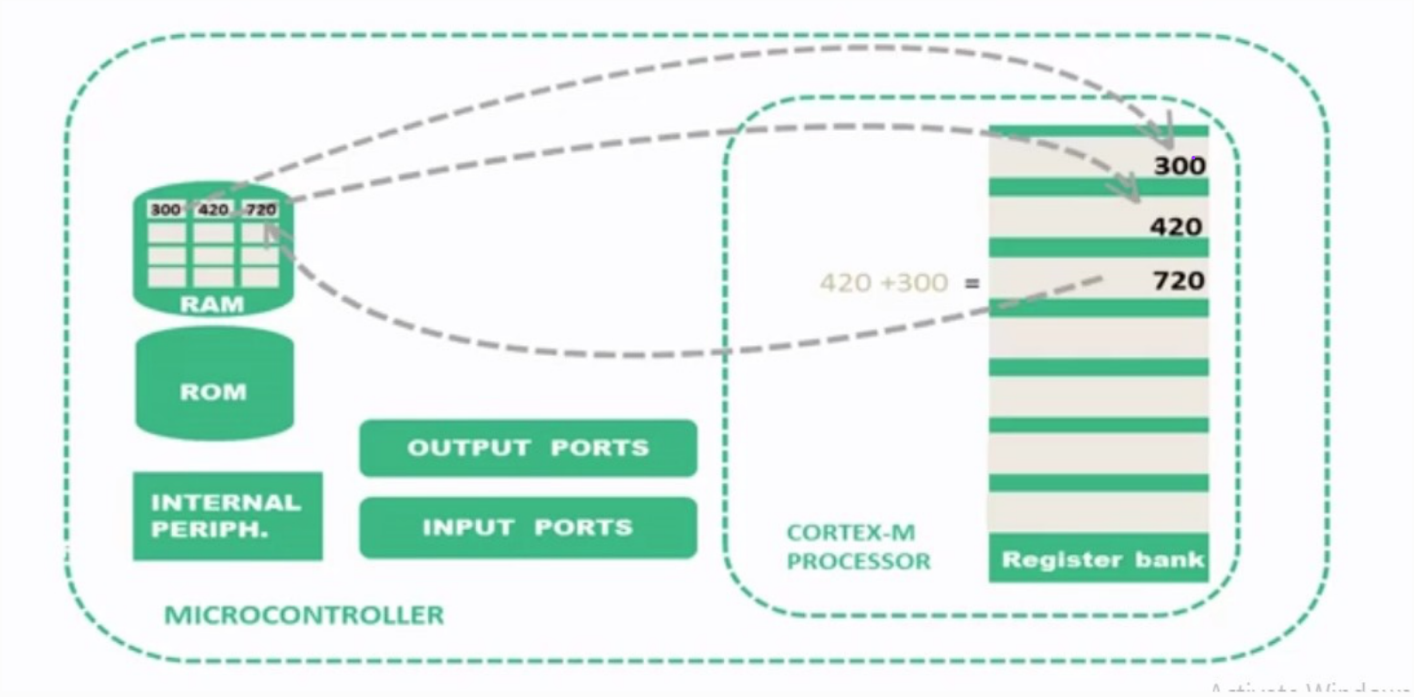
memory → Register transfer

Register process

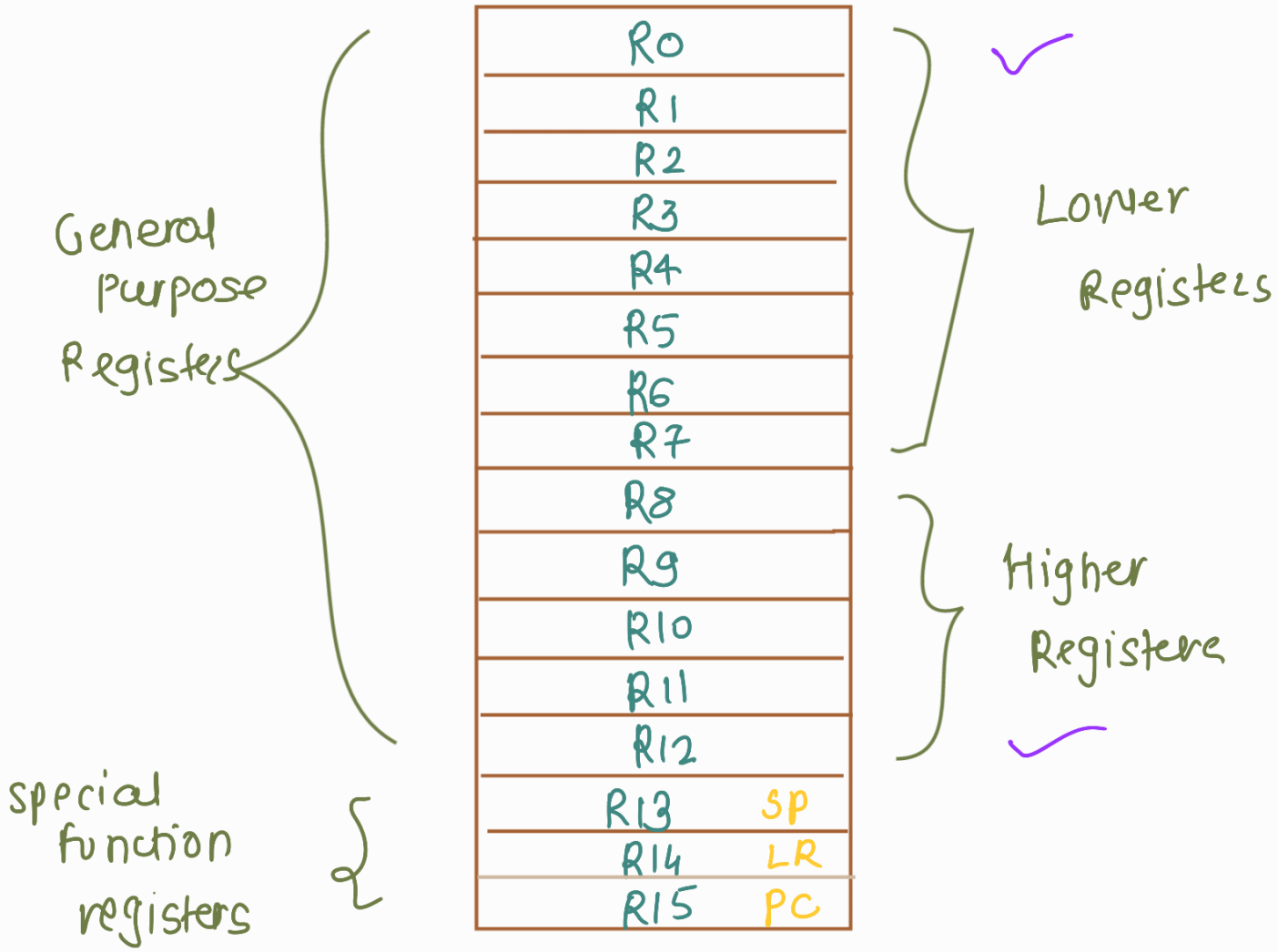
Register → memory

→ For example:- 300 + 420

300 & 420 stored in Memory



⊙ Register Bank



→ R0 – R12 ⇒ contain data and addresses

→ R13 ⇒ stack pointer & it points to top element of stack

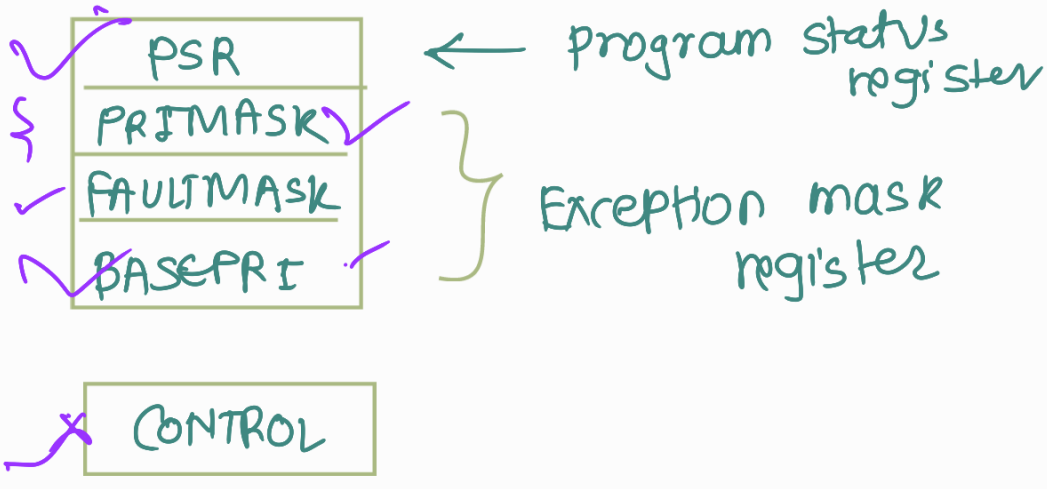
→ R14 ⇒ Link register and it is used to store the return location for function.

→ R15 ⇒ program counter, it is readable and writable

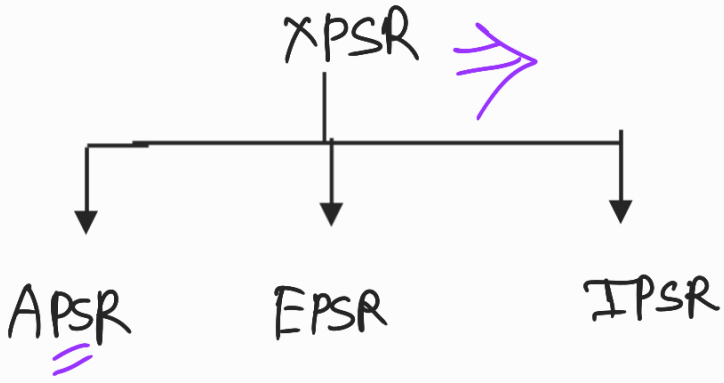
if you read : returns the current instruction address + 4

if you write : writing to PC causes branch operation.

⊙ Special Registers:



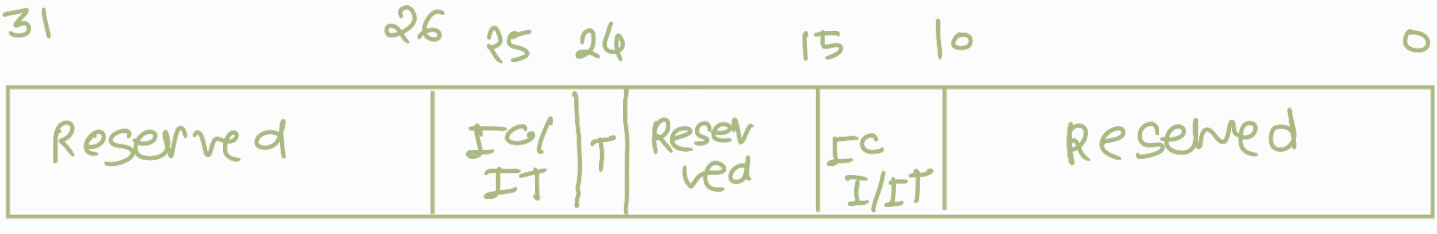
⊙ xPSR :



(i) APSR

(ii) IPSR

(iii) EPSR



Superimposition of those three registers make PSR register.

Instruction Set Architecture: (ISA)

→ All ARM Cortex-M processor are based on Thumb2 instruction which allows the mixture of 16-bits and 32-bit instructions.

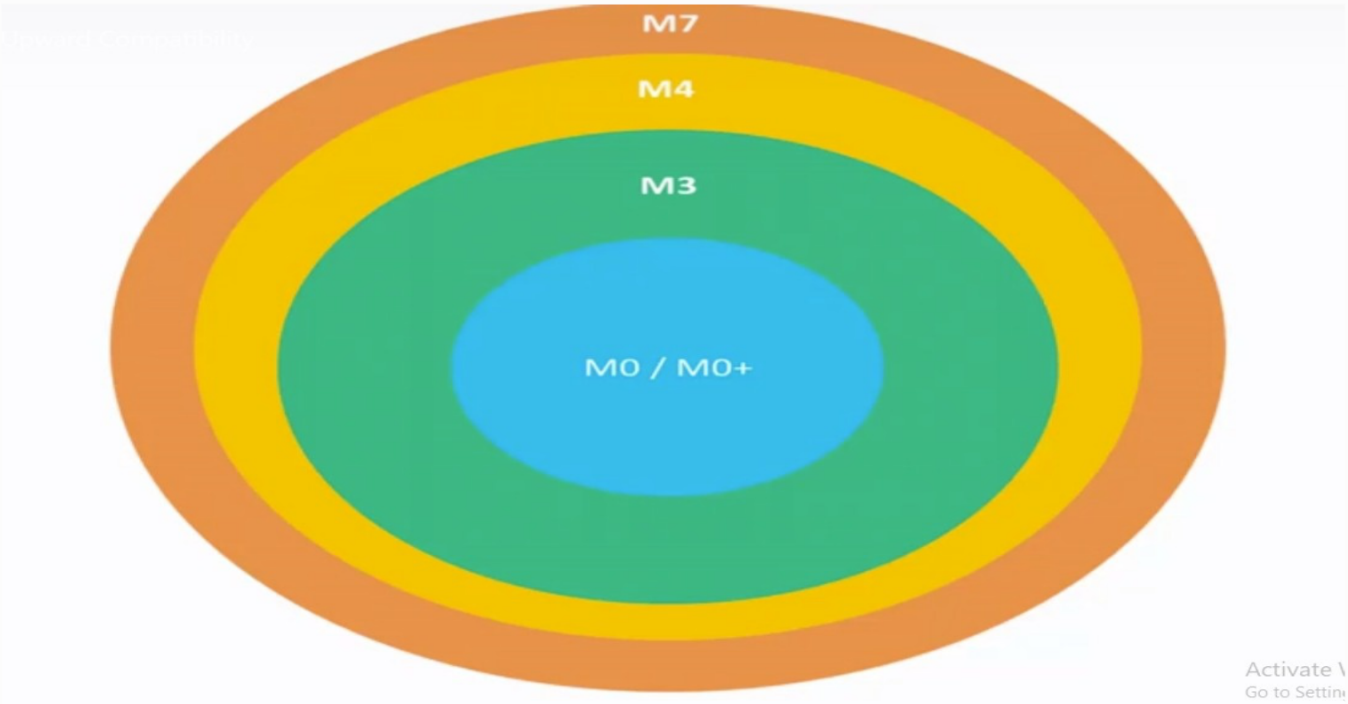
→ Arm - 32bit instruction set
  high performance
  low code density

→ Arm - 16 bit Instruction set
  high code density
  lack of performance

→ Thumb 2 Allows mixture of 16bit & 32-bit instruction set in order to have high perfo-zmance and code density.

⊙ Upward Compability :

⇒ Code compile for earlier version would run on next version.



⊙ Assembly Language Syntax :

Instruction have four fields separated by space or tabs.

Label	oprode	operand	; comment
init	Mov	R0, #150	; R0 = 150

- Label : optional. used to find the position of current instruction in memory.

- Opcode: specify processor operation to perform.

- operand: Specify source/destination of data to be processed by opcode

- comment: optional. used to explain code meaning.

Directive :-

→ They assit and control assembly process
→ They are also called pseduo -ops
→ They are not part of the instruction set
→ They change the way code is assembled

∵ Directive are Instruction used by Assembler to help automate assembly process and improve program modiability.

Examples: ORG (origin) ⇒ ORG 0000
EQU (equate) ⇒
DS.B (define space for bytes)

: Directives does not generate machine code. they are used essentially in preprocessing stage of assembly language.

(1) Thumb:

Placed at top of the file to specify that code is generated with Thumb instructions.

(2) CODE:

Denotes the sections for machine instruction (ROM)

(3) DATA:

Denotes the section for global variables.

(4) AREA:

instructs the assembler to assemble a new code or data section.

(5) SPACE:

Reserves a block of memory and fills it with zeros

(6) ALIGN:

Ensure next object allign properly

0x00 = 1 ← ORG 0x00
0x01 = 2 → MOV R0, #1
0x02 = 3 ← MOV R1, #2
0x03 =
0x10

Align 4
MOV R3, #3

(7) EXPORT:

To make object accessible from another file

(8) GLOBAL:

Same as Export.

(9) IMPORT:

To acess an "exported" object

(10) END:

Placed at end of each file

(11) DCB:

Places byte (8-bits) size constant in memory

(12) DCW:

Place a halfword (16-bits) size constant

(13) DCD:

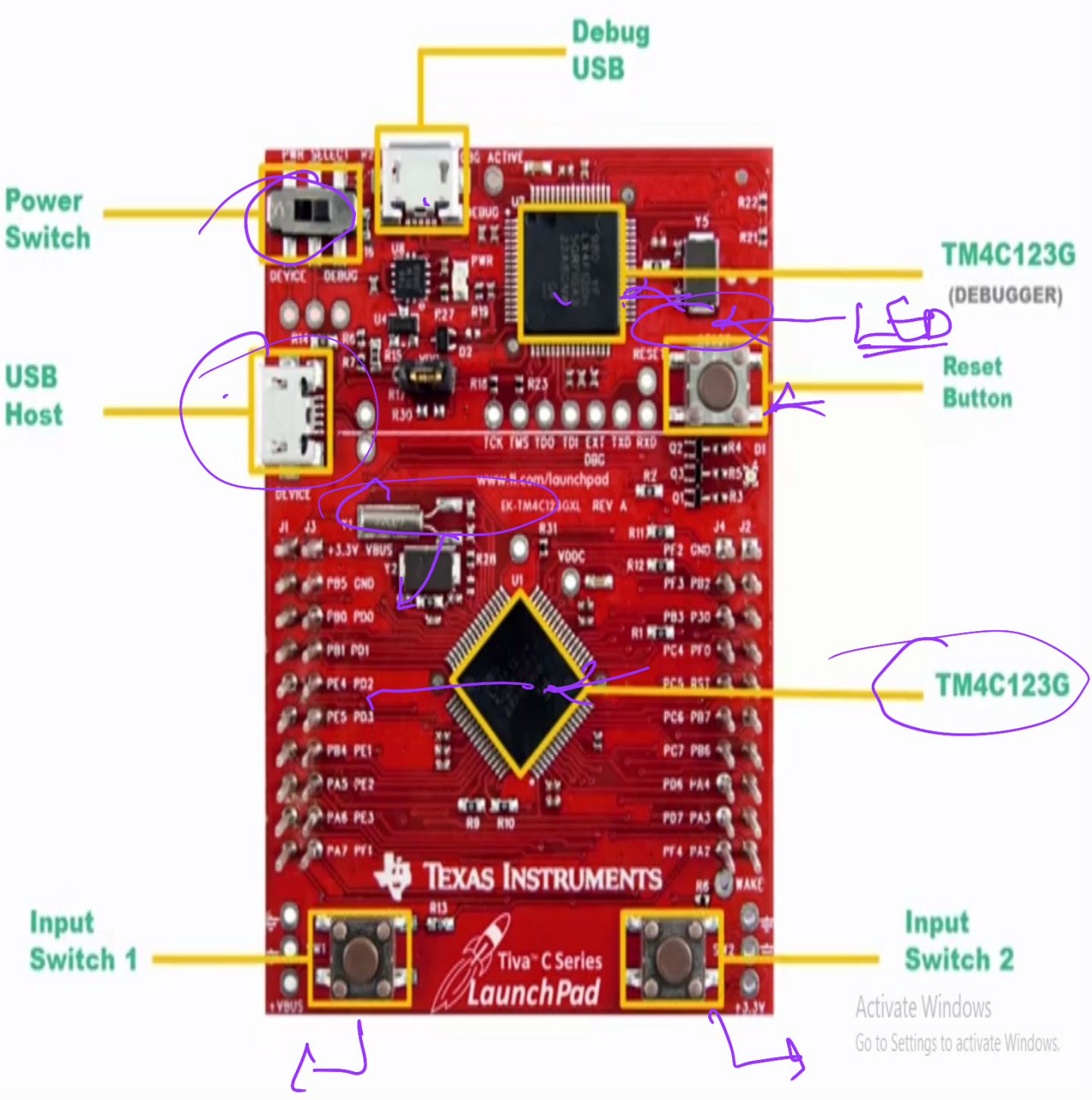
place a word (32-bit) sized constant in memory

(14) EQU:

To give symbolic name to a numeric constant

TIVA C LAUNCHPAD TM4C123GXL

- Processor = cortex M4
- flash = 256K
- Max speed = 80MHz



① project overview:

(1) Project ON - RED_ON

objective :-
To design, build and test the system that turns ON LED

Note:-

startup.c ⇒ C code

startup.s ⇒ Assembly code

→ set up starting point of program:
Reset handler

→ outlines the stack size

→ reset vectors and all interrupt vectors

Note:

→ Reset_Handler runs everytime the reset button is pressed.

MOV

Move data within processor without acessing memory.

Cannot MOV more than 8-bits value

Cannot MOV move to high register

Example	Example
MOV R0, #150 ✓	MOV R0, #256 ✗
MOV R1, R2 ✓	MOV R8, #3 ✗

LDR

→ Loads a registee with eigther: 32 bit constant value or a another registee.

→ Reads 32-bit value from memory to registes

Example:

LDR R0, =0XFFFF ✓

LDR R1, [R2] ✓

STR

→ Stores a register value in memory

```
STR   R0, [R1]
```

R0: Source
R1: destination

B

→ Branch to Location

Example:

```
B    loop          ; branch to / jump to loop
```

BL

→ Branch to subroutine

```
; call Turn_on , Turn_on is a subroutine
BL   Turn_on
```

BX LR

→ Return from subroutine called

```
BLUE ON  ; subrounne
MOV   R2, #0X01
BX    LR
```

Nop

→ No operation, do nothing

Example:

```
STR   R0,[R1]   ; Store value of R0 to R1
NOP
NOP
```

} do nothing for 2 clock cycles

→ 1 Nop = 1 cycle

Aim: TO turn on RED LED After switch press

OPCODES:

1) MOV
2) LDR
3) STR
4) B
5) BL
6) BX LR
7) NOP

CMP

→ Subtracts the value of the second operand from first

→ Example:

Compare the value of R0 with # 0X01

CMP R0, #0X01

= APPR N, Z, C;
= Z = Not Set
= Z = Set

BEQ

Branch if equal to 0

→ Mostly used with CMP instruction

→ Example : (Z flag)

```
CMP   R0, #0x01
BEQ   LED1
```

BNE

Branch if not equal to 0 (!=0)

→ check Z flag

→ Example;

```
DELAY :
  LDR R0, = ONESECOND
  SUBS R0, R0, #1
  BNE  DELAY
```

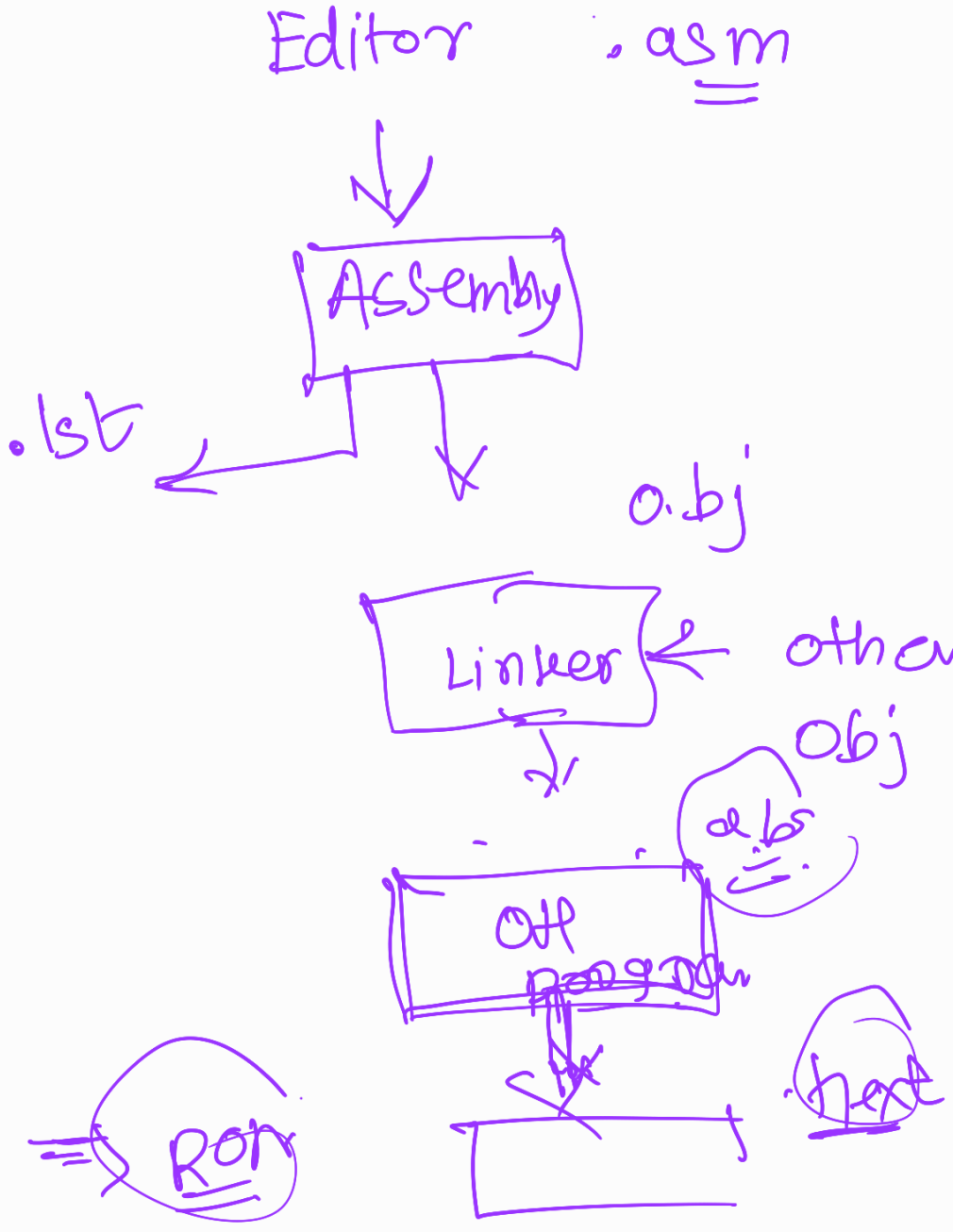
SUBS

SUBtract and set Flags in APSR

→ Example :

```
; R0 = R0-1 , set Flag in APSR

SUBS R0, R0, #1
```


Editor
.asm
Assembly
.lst
o.bj
Linker
other
Obj
abs
OP
text
ROM

Switch configure = port f 0, 4

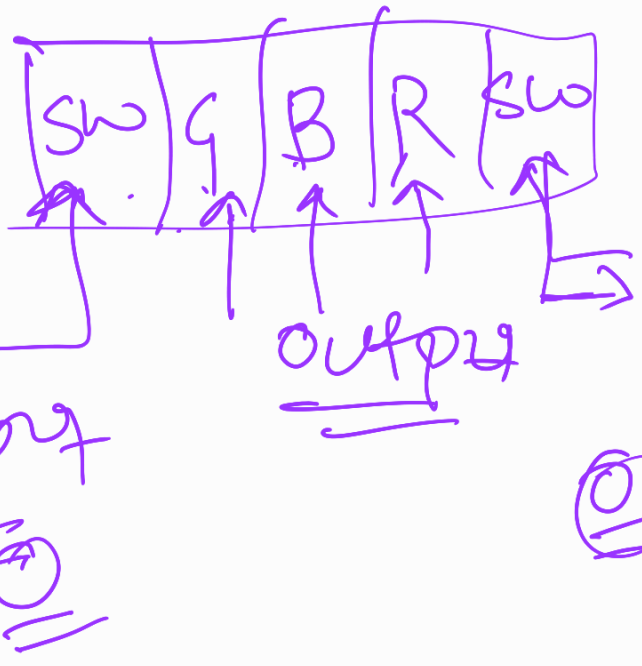
LED ⇒ 1, 2, 3,

Switch press = LED – ON

Switch press = LED – OFF

PUR

pd



RO = 0 = 0X00

RO = 1 =