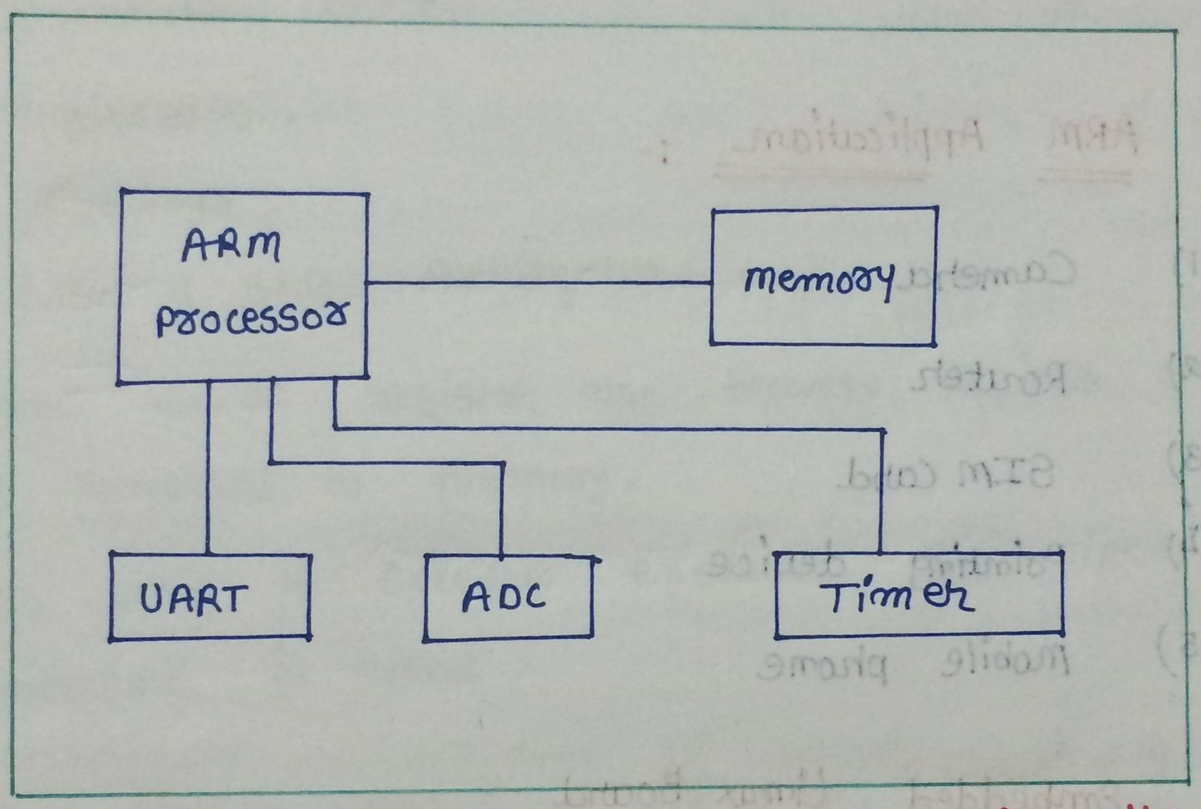
# ARM Processor :-

→ It is a processor.

→ designed by ARM limited OR ARM holding.
(Latest Name OF ARM Limited).




ARM based SOC
↓
(System on chip)

ARM µC

## ⊛ SOC Manufacture :-

1) Qualcomm
2) Samsung
3) Broadcomm
4) Nxp semiconductor
5) Atmel

## SOC'S :-

1) Snapdragon (by Qualcomm)
2) BCM2708 (by Broadcomm)
3) LPC 2148 (by Nxp Semiconductor) (Vector board)
4) AM335X (by Sitara)

## ARM Application :-

1) Camera
2) Router
3) SIM card
4) Printing device
5) mobile phone

## Embedded Linux Board (Single board computer) :-

1) Raspberry Pi (BCM2708)
2) Beaglebone black (AM335x)
   OR
   Beaglebone white (AM335x)
3) Cubie board
4) Friendly ARM

etc

→ ARM processor is a RISC type processor.

⇒ RISC type CPU is Fixed length instruction.

⇒ In RISC mostly single clock cycled instruction available.

## ★ 4 Rules OF designing RISC Based CPU :-

1) Instruction set.
2) Register's.
3) Pipelings.
4) Load & store Architecture.

⇒ More no. OF register use because register access Faster compared to memory.

⇒ In order to speedup execution them piped architecture is used.

**Question** Pipline How it is speedup execution?

→ pipeline speedup execution by fetching next instruction while other instruction are being declared or being executed.

| | Fetch | Decode | Execute |
|---|---|---|---|
| cycle1 | ADD | | |
| cycle2 | SUB | ADD | |
| cycle3 | cmp | SUB | ADD |

| | |
|---|---|
| CMP | 0x03 |
| SUB | 0x02 |
| ADD | 0x01 |
| M/R | 0x00 |

ARM instruction take 1 clock cycle to execute.

→ only 1st instruction needs 3 clock cycled to execute while the Host followed by it, needs only one (1) clock cycled to execute.

→ Same processor is used Instruction Queue used for Speedup the execution.

ex:- 8086, Intel

→ Some processor is use pipeline Architecture for Speedup the execution.

→ all RISC Based µp → pipeline Architecture.

## ⊛ 3 pipeling (when it fails) :-

1) during the execution of loops.

```
while (condition)
{
    1st
    2nd
    3rd
    4th
}
    1st
    2nd
    3rd
```

| | Fetch | Decode | execute |
|---|---|---|---|
| | Cond$^n$ | | |
| | 1st | Cond$^n$ | |
| | 1st | Cond$^n$ | Cond$^n$ |

⇒ If it is Failed the pipeline is cleared.

## ⊛ Exception when pipeline will be Flused :-

1). looping condition.
2) interrupt occurs.
3) ISR is Called.
4) If ---- else Statement.

⇒ In RISC only Load & store instruction can access the memory that reduce memory access time.

→ In RISC You get a seprate set of load & store architecture instruction to internal with memory.

## ⊛ Load and Store Architecture :-

only load and store instruction can Access the memory.

⇒ which reduces memory access time.

⇒ since micro processor will interact with registor itself RISC based processing is known as "Register Intensive processing".

## ⊛⊛⊛ ARM 7 Architecture Features :-

1) Low power, High performance processor (work on 3.3 V)

2) ARM working Frequency is 60 MHz.

3) 1 Machine cycle 1Q instruction execute.

4) ARM7 core uses a RISC Architecture & Von-numann type bus architecture.

5) ARM7 is 32 bit processor core (CPU).

6) ARM7 supports 3 stage pipeline architecture.

3 Stages are

- Fetch
- Decode
- Execute

7) Reduce gate count design, suitable for small, low power, high performance implementation.

8) Die size is very less or very small.

9) ARM 7 Core supports on chip Hardware debug Technology.

JTAG Technology.

→ After downloading the program we can debug the program step by step using JTAG Cable.

64 pin :-

→ 16 - 20 pin is reserved for Hardware debugging.

10) uniform and Fixed length instruction set to Simplify instruction decode.

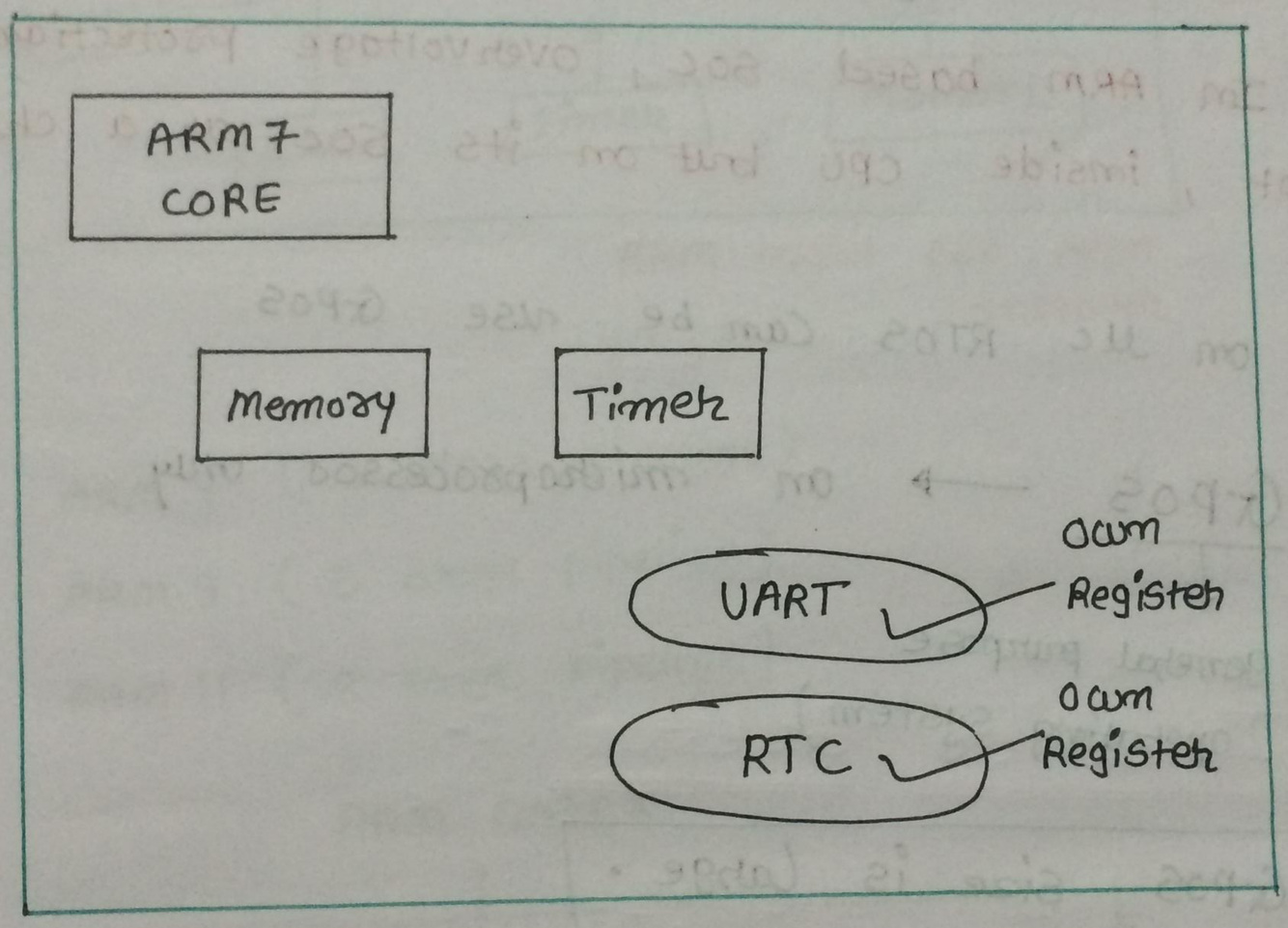
11) Load and Store multiple instruction to maximise data throughput.

12) ARM7 core supports 37, 32 bit registors

16 registor are available for uses.

→ Each phetiphetal has minimum 10 registors.

10)




ARM system Devlopes Guide by Andrew Sloss
(only theory information)

LPC214 X. user Manual
(For practical information)

13) ARM7 core supports 7 mode of operation.

* user } → non-previledged mode
(user can't write into the some Register).

* system
* supervisor
* FIQ
* IRQ
* Abort
* undefined

} previledged modes
(user will be able to write on some Registor)

14) In ARM based SOC, overvoltage protection ckt not, inside CPU but on its SOC as a chip.

⊛ on µc RTOS can be use GPOS

GPOS → on microprocessor only
↓
(general purpose operating system)

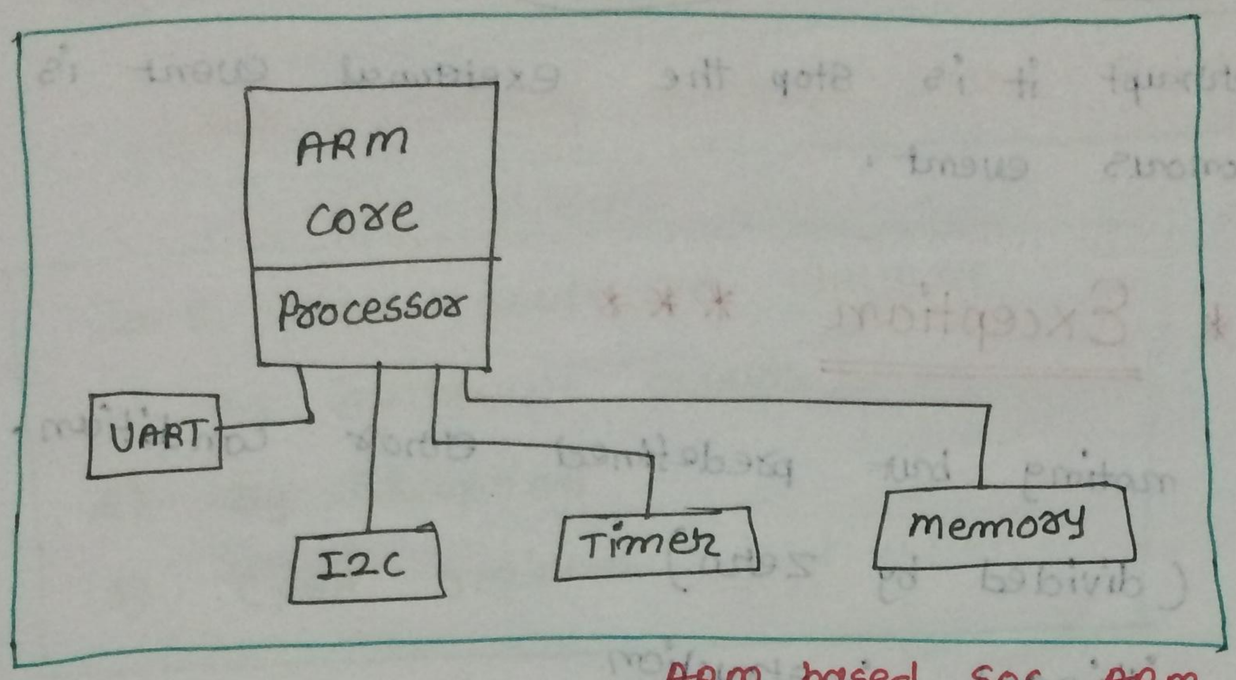
| | |
|---|---|
| GPOS | size is large. |
| RTOS | size is small. |

→ In mobile phone 98 to 99 % use ARM cortex.

ARM cortex (32 bits and 64 bits).

ARM 7 → Only Study purpose

ARM 9
ARM 11
ARM Cortex




ARM based SOC. ARM controller

ARM 7 ( 3 Stage pipeline)

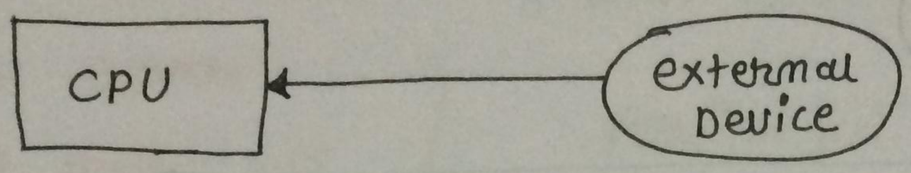
ARM 9 ( 5 Stage pipeline)

ARM 11 ( 8 Stage pipeline)

ARM Cortex

- Series A — in mobile phone
- Series M — microcontroller
- Series R — Real Operating System

⇒ when Interrupt or Exception are come than ARM mode is changed.




interrupt it is stop the external event is Asynchronous event.

## *** Exception ***

⇒ It is noting but predefined error condition.

eg :- (divided by zero)

- illiegal instruction
- Segmentation Fault.

⇒ due to interrupt or due to exception go to the any mode of 7 mode.

## Interview Question :-

### Explain about mode :-

① user mode :-

⇒ Normal program execution state.

⇒ upon reset, startup code is execute and it will clear the memory Register.

Note :- This mode can only be changed when an interrupt or exception is generated.

→ Through instruction, processor change the mode of exception then user has to modify CPSR register.

| |
|---|
| CPSR = current program Status Register. |

| |
|---|
| In 3 case mode are changed. |
| 1) By interrupt occurs |
| 2) By exception |
| 3) CPSR register. |

⇒ in uses mode CPSR is read-only register. can't be change it.

In user mode

mov CPSR, #data

⇒ Assembler won't generate error but if won't be able to modify CPSR.

## (2) System mode

→ privileged mode of uses.

Startup code

↓

main()
{
}

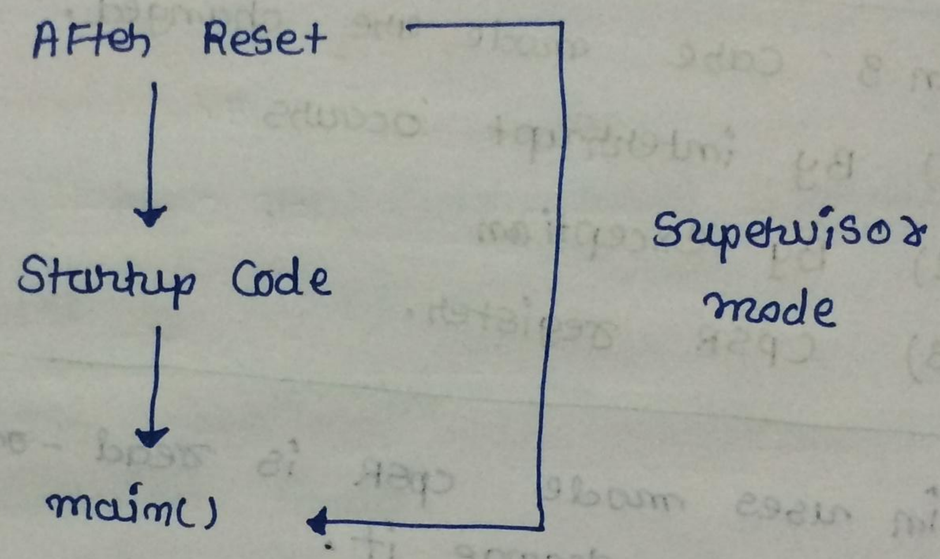
user mode

↓

how to go to system mode
(with using CPSR bcoz CPSR can't be accessed in user mode.)

## ③ Supervisor mode

⇒ mode of core after reset.




⇒ After reset and Before main() processor will be in a Supervisor mode.

⇒ Default mode of ARM processor.

⇒ after reset the µp then Startup code is Run in a Supervisor mode.

⇒ only in Supervisor mode MSR instruction will be there using which we can change the mode by modifying CPSR.

## ④ FIQ mode :-

→ Fast Interrupt Request.

→ Low Latency interrupt Service.

Low Latency :- means CPU response to interrupt is Faster.

ARM7 TDMI

- ARM7 → Core processor name
- T → H/W debug Technology
- D → MAC unit
- M → Interrupt.

## ⑤ IRQ mode :-

→ Interrupt Request.

⇒ Compare to FIQ mode IRQ latency is more.

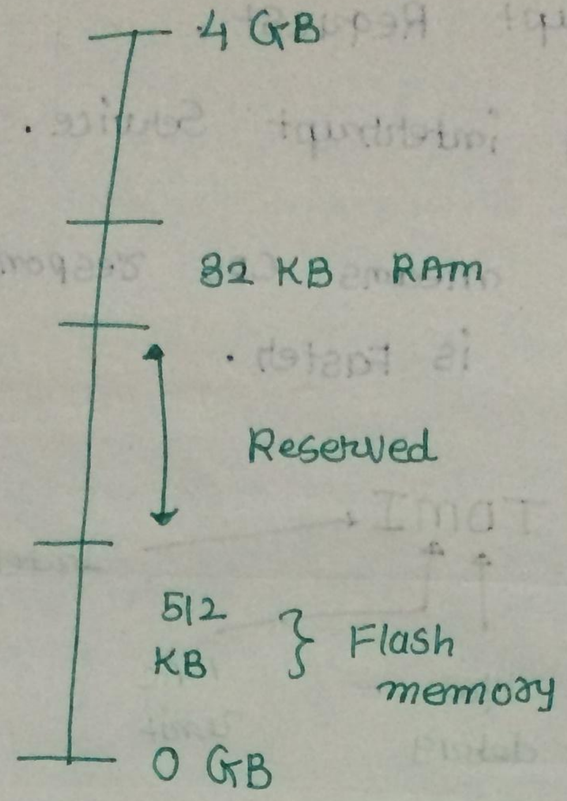
## ⑥ Abort mode :-

⇒ This mode is selected when data or instruction fetch is aborted.

⇒ ARM processor attemps to Fetch an instruction From an address without the correct access permission than processor goes to the Abort mode.

| |
|---|
| Size of ARM program Counter = 32 bit |

LPC2148




## (7) undefined mode :-

⇒ This mode is selected when undefined instruction is fetched

e.g. illiegal opcode (illegal instruction)

⇒ Then up goes to this undefined mode.

## (*) ARM7TDMI - S Processor :-

ARM7 = Family name

T = THUMB Instruction set

D = Hardware Debug Technology

M = MAC Unit

I = Interrupt (FIQ)

S = Synthesize

## ⊛ THUMB Instruction mode :-

→ It is also called a 16 bit instruction set.

| Family | Architecture | Instruction set support. |
|---|---|---|
| ARM7 | armv3 | Only 32 bit instruction set. |
| ARM7 | armV4T | 16 bit & 32 bit instruction set. |

| | | | |
|---|---|---|---|
| 16 bit instruction set | = | THUMB instruction set. |
| 32 bit instruction set | = | ARM instruction set. |

## ⊛ processor Operating Status :-

1) ARM State
2) THUMB State

| ARM State | THUMB State |
|---|---|
| Processor execute 32 bit word aligned ARM instruction. | processor execute 16 bit half word aligned instruction. |

64 bits = double word

32 bits = word

16 bits = Half word

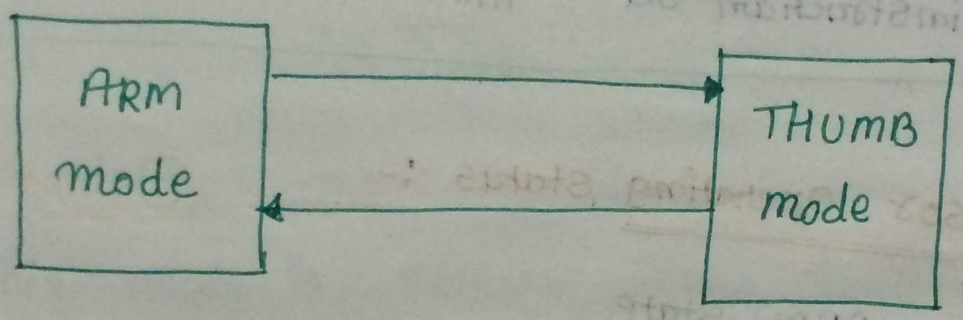
8 bits = byte

4 bits = Nibble.

⇒ upon Reset ARM processor

- In supervisor mode
- ARM instruction (32 bit instruction)

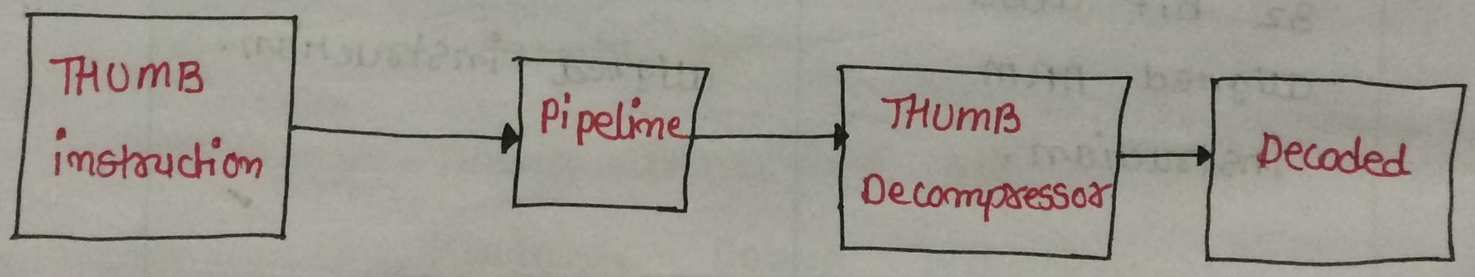
| |
|---|
| In ARM7 Instruction |
| ARM Instruction - 51 |
| THUMB Instruction - 81 |




⇒ THUMB instruction are dynamically decompressed to ARM instructions.

ARM → UART program size is large.

THUMB → UART program size is small.




## Advantage OF THUMB :-

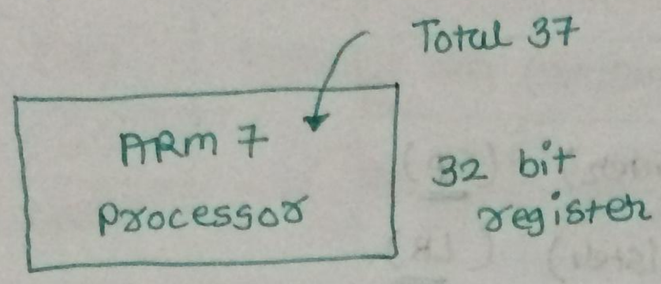
1) Reduce the code size.

⇒ Hardware synthesize using software (VHDL).

⇒ Using Bx instruction to change the state.

- ARM to THUMB
- THUMB to ARM

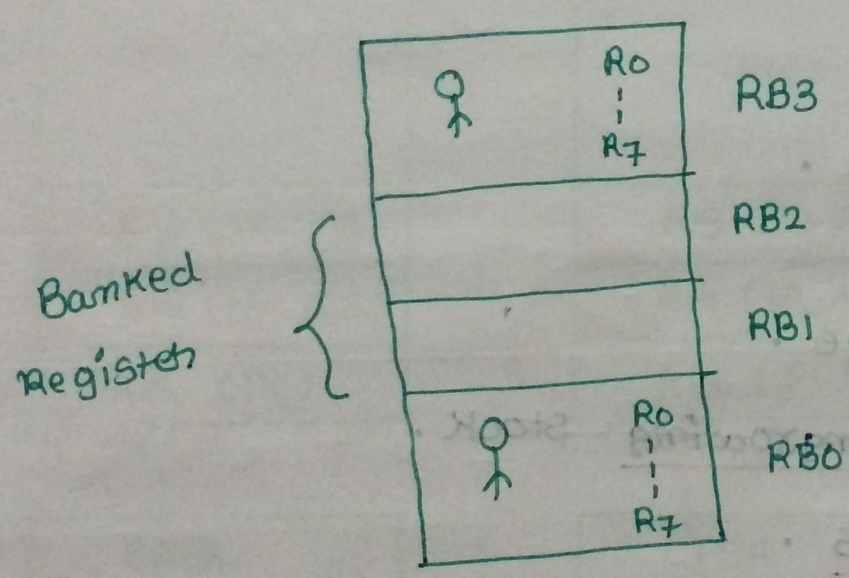
⇒ In ARM9 8 mode is available.




16 bit Register are used for user.

* when processor is in USER mode OR System mode than use 16 Register available.

⇒ Invisible Registers are called as a "Banked Registers"




In 8051 RAM

⇒ In ARM 16 Register are available for user and remaining 21 registers is called as a "Banked registers."

$$\begin{array}{r} 37 \\ -16 \\ \hline 21 \end{array} \longrightarrow$$ Banked Regs. (Invisible Register).

# *** ARM7 Register set ***

R0
R1
R2
⋮
R12 } General purpose Registers

Special Function Registers:
- R13 (Stack pointer) (SP)
- R14 (Link Register) (LR)
- R15 (Program Counter) (PC)

⇒ all registers of ARM7 can hold the data or address.

eg:- MOV R0, #data is support in ARM.

⇒ Return address is store into Link register.

3 case :-

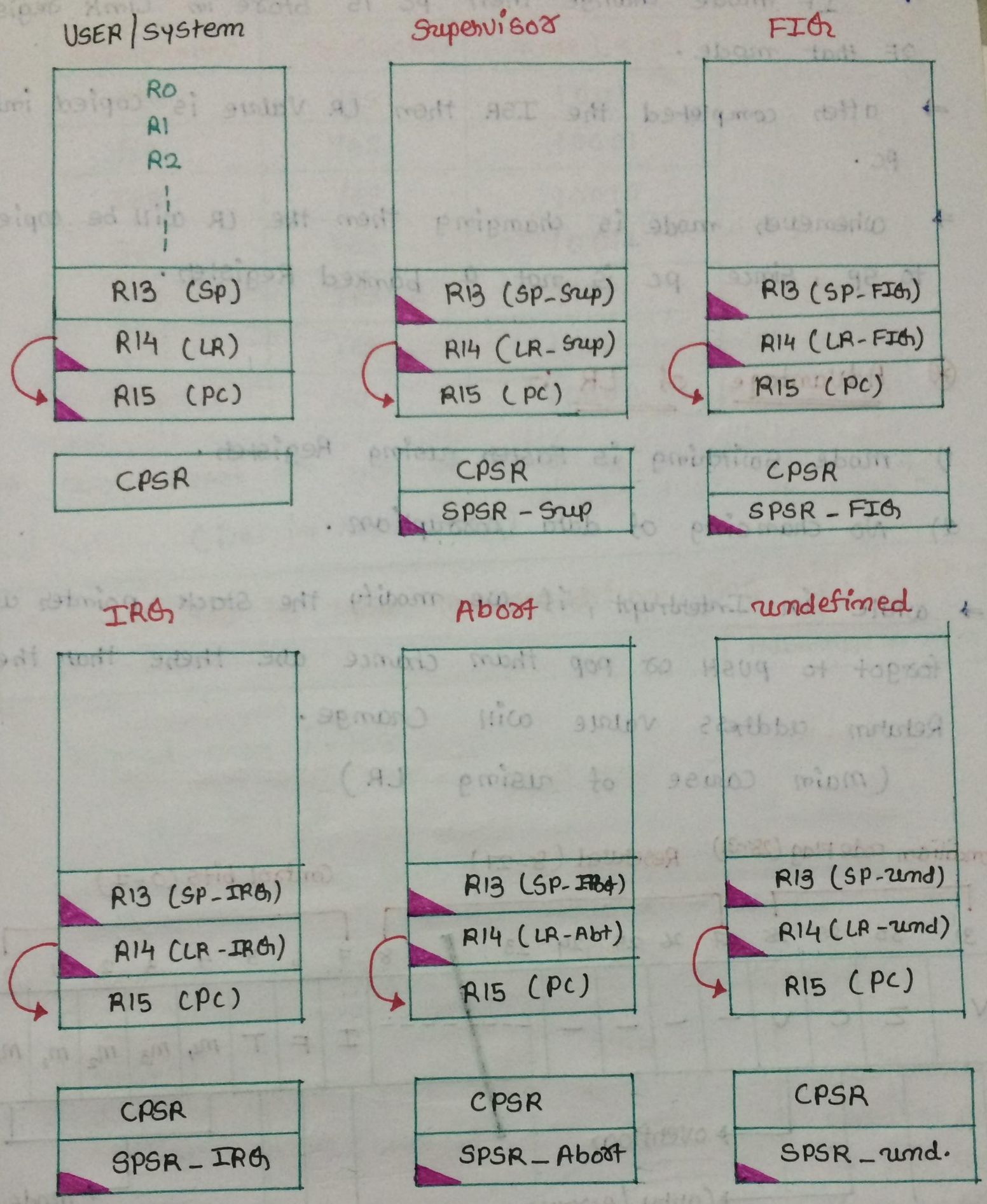
1) Function call.
2) Interrupt.
3) Mode change.

⇒ ARM7 Stack is downgrowing stack.

⇒ Stack size = 4 bytes.

⇒ with every push instruction the sp will be decremented by 4 bytes.

⇒ Instruction to write into program counter is present in ARM7.




→ Thus $\boxed{\text{SP and LR}}$ → banked Registers

$\boxed{\text{CPSR and PC}}$ → not banked Registers

⇒ IF mode change then pc is store in Link register OF that mode.

⇒ after completed the ISR then LR Value is copied into PC.

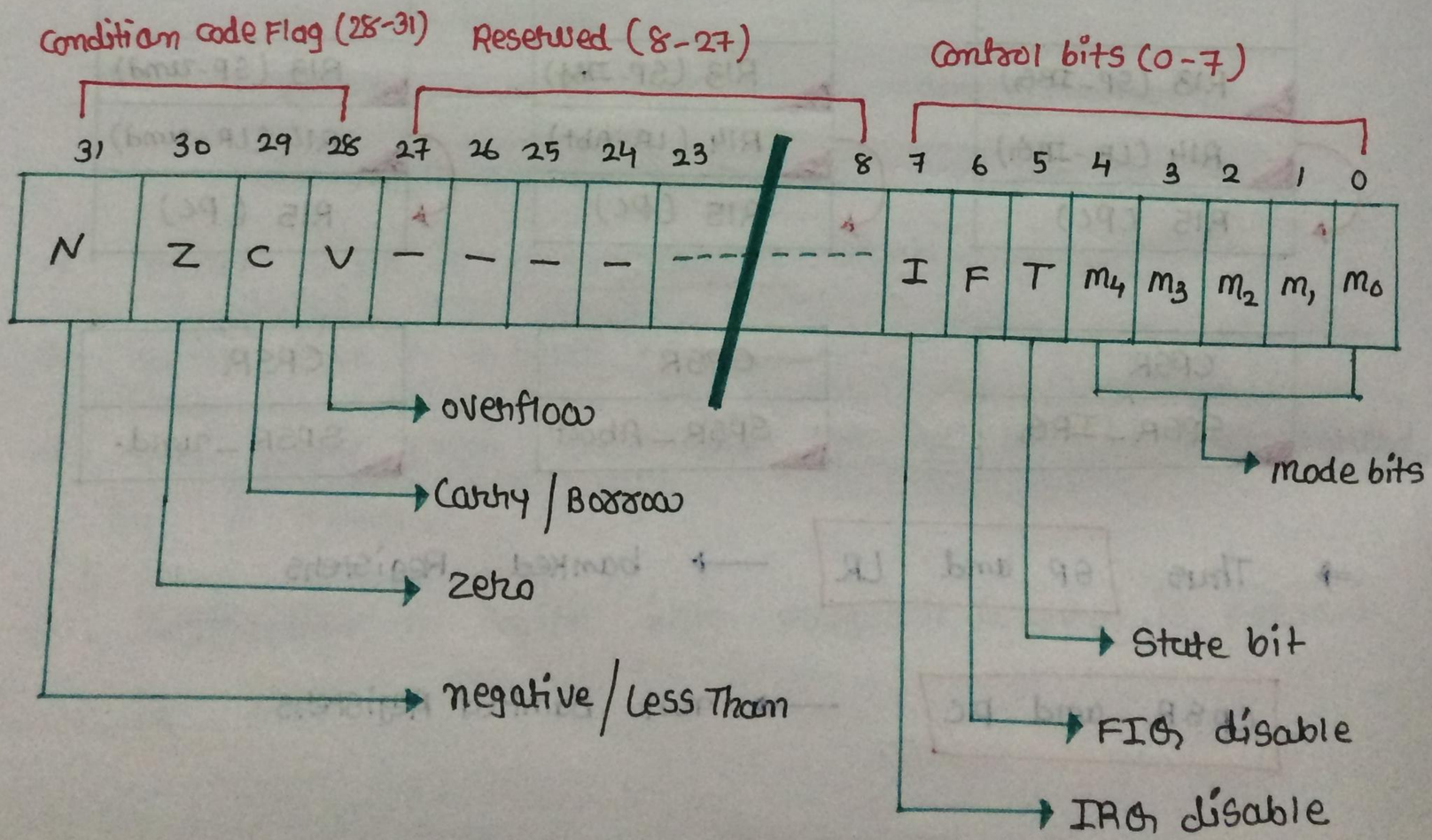
⇒ whenever mode is changing then the LR will be copied to sp, since pc is not a banked Register.

## ⊛ Advantage of LR :-

1) mode switching is Faster using Register.

2) No chancing of data corruption.

⇒ while in Interrupt, if we modify the stack pointer and forgot to PUSH or pop than chance are there that the Return address value will change.
(main cause of using LR)



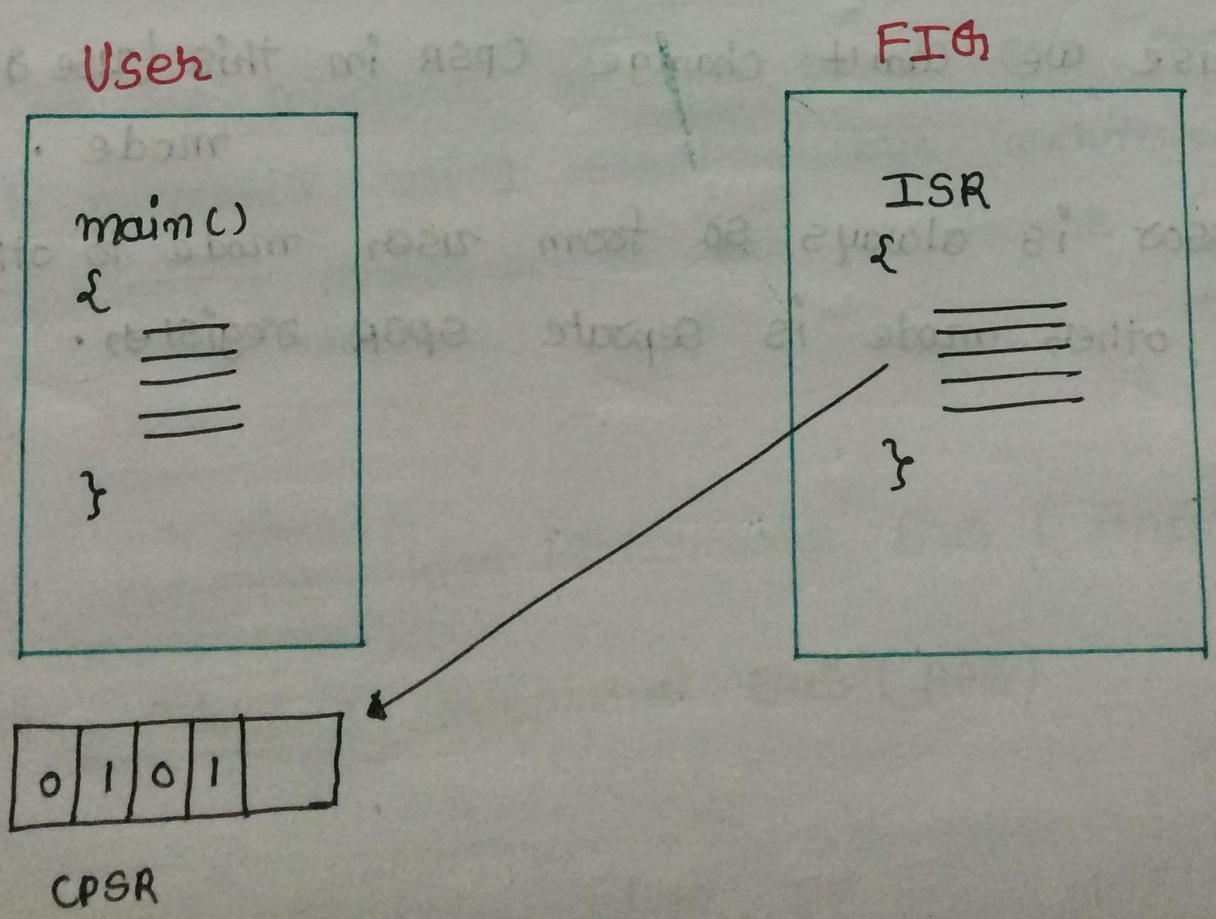
| Abbreviation | Privileged | Mode [4:0] |
|---|---|---|
| abt | Yes | 10111 |
| fiq | Yes | 10001 |
| IRQ | Yes | 10010 |
| SVC | Yes | 10011 |
| SYS | Yes | 11111 |
| Und | Yes | 11011 |
| usr | NO | 10000 |

⇒ upon reset 5th bit of CPSR is 0.
(i.e in ARM state)

upon divided by zero than overflow flag will be set.

Advantage of ARM

signed char i = 128
-128 to 127
then OV Flag will set.




arithmatic operation perform in ISR that time CPSR changed.

Que :- How to solve it ?

⇒ Internally when mode is changed the CPSR will be copied into SPSR after completing the mode operation again SPSR will be copied back into CPSR there by avoiding the misbehaving.

⇒ SPSR is banked Register for all mode except user mode.

ie. every mode except user mode have its own copy of SPSR register.

| SPSR = Saved Program Status Register. |
|---|

Interview Que :- why SPSR is not available in User mode ?

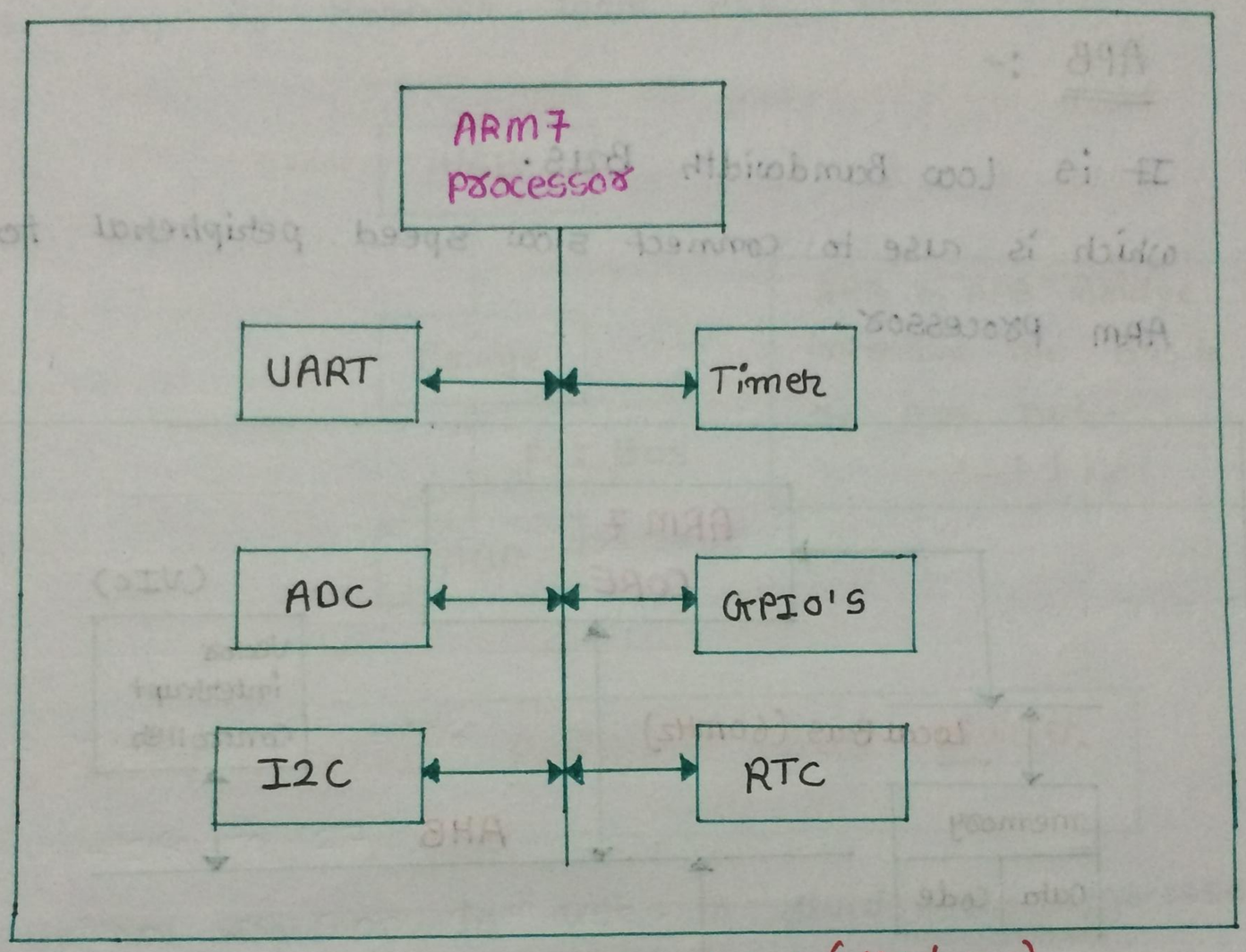
⇒ CPSR ⟶ SPSR (when mode change)

Because we can't change CPSR in this (use & sys) mode.

⇒ processor is always so from user mode to other mode so other mode is seprate SPSR register.

# Advance Microcontroller Bus Architecture :-

## (AMBA Bus Architecture)




ARM Based (µc / soc).

⇒ on chip peripheral of ARM based soc are connected to processor using standard bus architecture called advanced microcontroller bus architecture. (AMBA)

## ⊛ Types of AMBA :-

1) Advance high performance Bus (AHB)

2) Advance peripheral Bus (APB)

① **AHB :-**

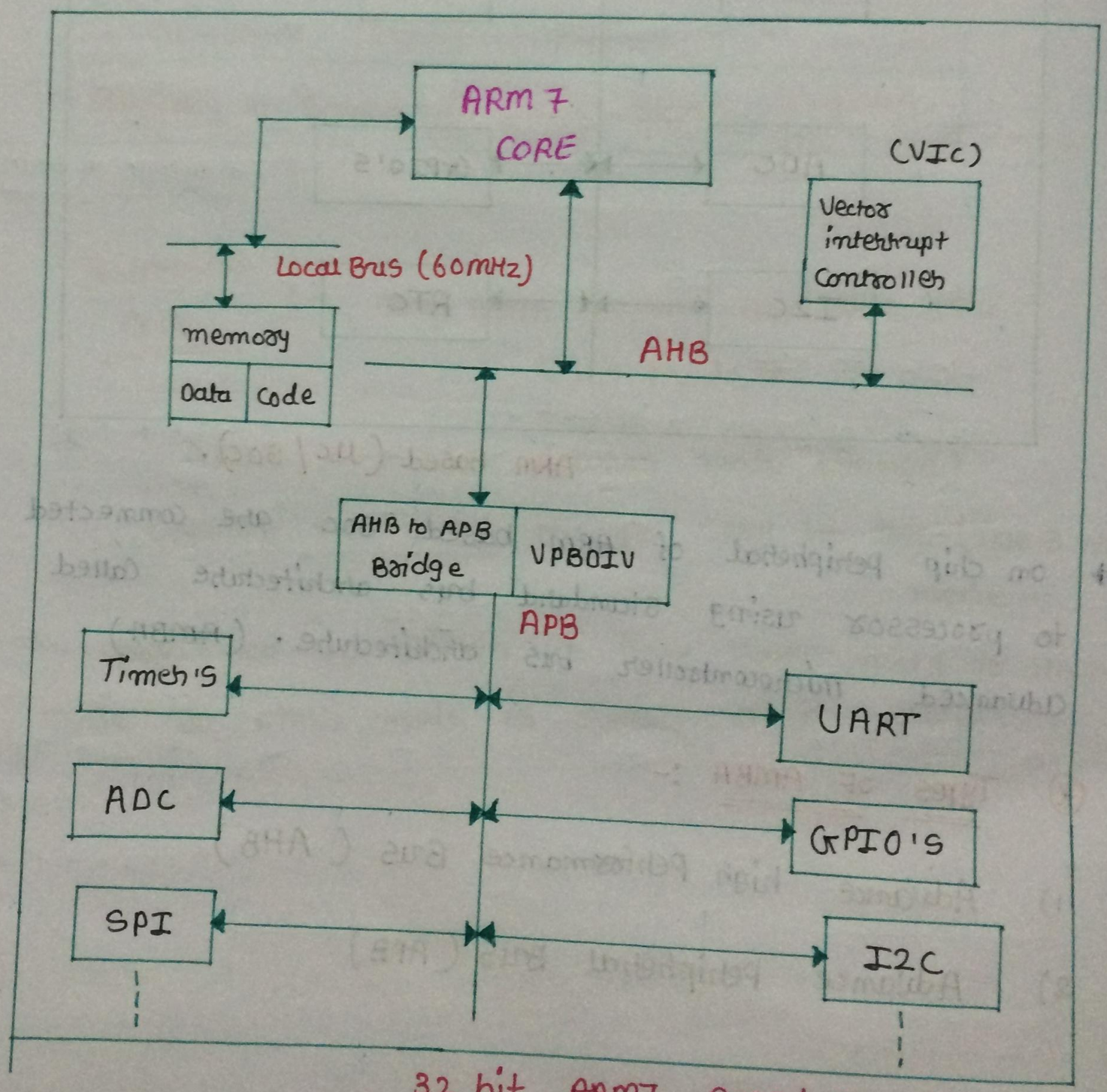
⇒ It is a high bandwidth Bus.

⇒ It is use to connect fast speed peripheral to RAM Processor.

② **APB :-**

⇒ It is low Bandwidth Bus.

⇒ which is use to connect slow speed peripheral to ARM processor.



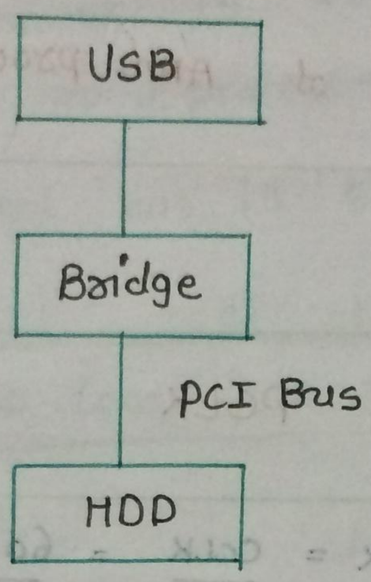

32 bit ARM7 Based SOC (µC)

## PCI BUS :-

Peripheral Componet Interconnect.

use on a motherboard ( It is a Standard Bus architecture

→ when connect the USB in a laptop than data of USB is copy in Harddisk than PCI Bus is work.




AHB to APB Bridge interface the Bus to the AHB Bus.

**VPBDIV = VLSI Peripheral Bus Divider**

→ VPBDIV Register is use to divide the processor frequency and generate the peripheral clock using processor clock.

ARM7 core = 60 MHz

its called a processor clock (CCLK)

↓

"Core clock"

**ARM based SOC processor Clock**

**CCLK = 60 MHz**

⇒ with the help of VPBDIV processor frequency is divided and peripheral frequency is generated using PCLK ( Peripheral clock )

| |
|---|
| PCLK is frequency of peripheral. |
| CCLK is frequency of ARM processor. |

| VPBDIV | PCLK |
|---|---|
| 0 | $PCLK = \frac{CCLK}{4} = \frac{60\,MHz}{4}$ <br> $\therefore PCLK = 15\ MHz$ |
| 1 | PCLK = = CCLK |
| 2 | $PCLK = \frac{CCLK}{2} = 30\ MHz$ |
| 3 | reserved |
| 4 | $PCLK = \frac{CCLK}{4} = 15\ MHz$ |

⇒ VPBDIV = 0

⇒ The VPBDIV value is set in startup code.

⇒ when we change VPBDIV value VPBDIV = 1 then change the value of PCLK.

→ modify the VPBDIV than change the value of PCLK.

## VIC (Vector Interrupt Controller) :-

⇒ all the interrupts (FIQ, IRQ) are handle by Vectored Interrupt Controller.

⇒ External controller is required for to control the interrupt in a micro processor.

⇒ Speed of local bus is same as a ARM processor clock means

| Speed of Local Bus = 60 MHZ |
|---|

## *** Features of LPC 2148 ***

SOC → LPC 2148

CORE → ARM7 TDMI-S

ARCH → arm V4T

① ARM7 TDMI-S based high performance 32-bit RISC microcontroller with THUMB extensions.

② 512 KB on-chip Flash Rom with In system programming (ISP), In application programming.

→ allow to modify the data of Flash memory at runtime.

③ 32 KB RAM

④ Vectored Interrupt controller.

→ It's handle 32 IRQs.

⑤ Two 10 bit ADCs with 14 channels.

⑥ USB 2.0 Full speed device controller.

⑦ Two UARTS (UART 0 & UART 1), one (UART1) will Full modem interface.

⑧ Two I2C serial interfaces.

Out of 9 pin's In UART. You use Tx, Rx, and GND its called "NULL modem" or "half modem."

If you use all 9 pin's its called a "Full modem"

⑨ Two SPI serial interfaces.

⑩ Two 32-bit timers.

⑪ watchdog Timer.

⑫ PWM unit.

⑬ Real time clock with optional battery backed.

⑭ Brown out detect circuit

→ It is protect the processor from over Voltage.

⑮ General purpose I/O pins (GPIO's).

LPC2148 GPIO'S

- PORT 0 (32 I/O pins)
- PORT 1 (16 I/O pins)

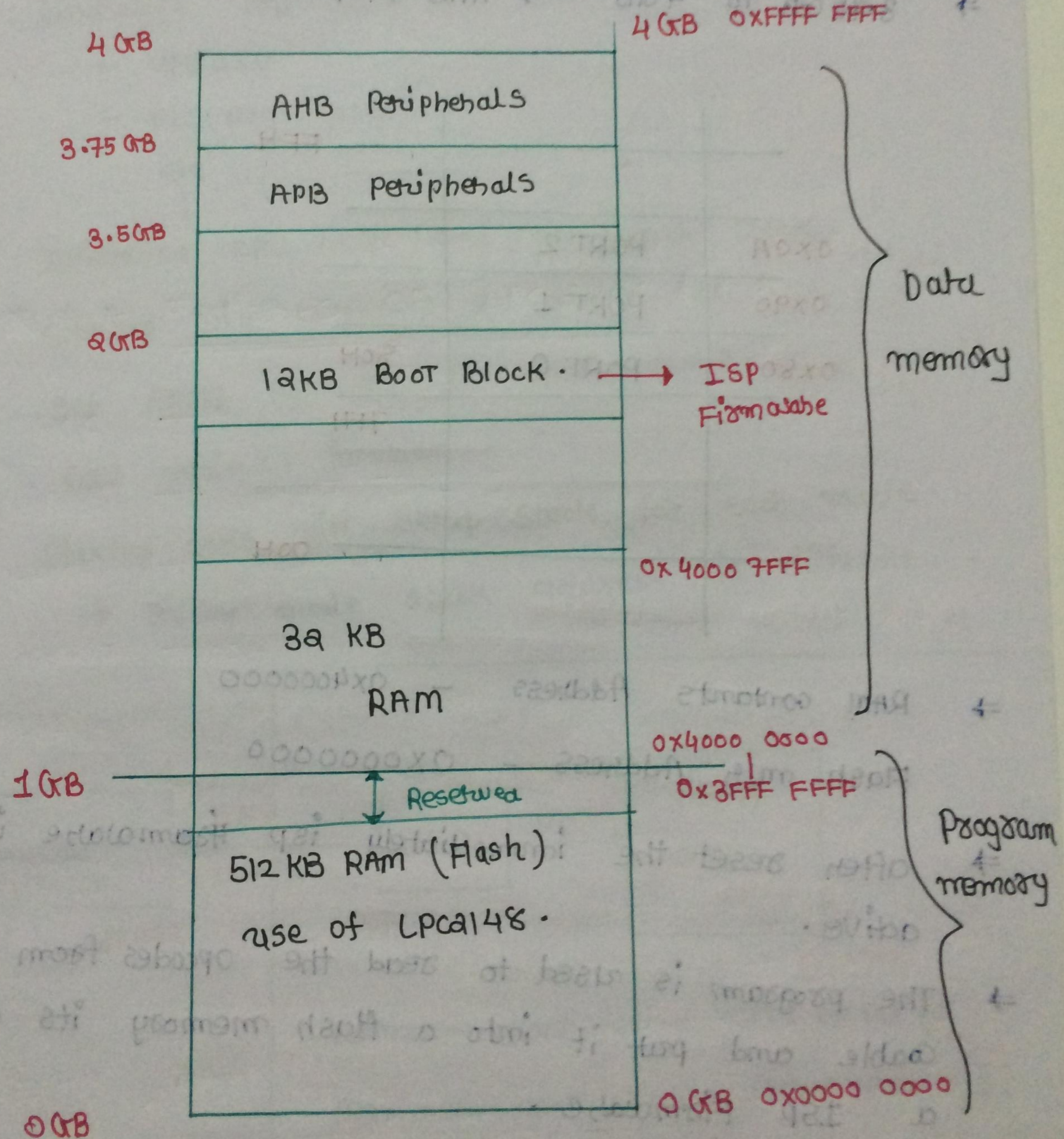
(16) CPU Clockup to 60 MHz.

(17) on-chip crystal oscillator.

(18) on chip PLL

↓

(Phase Lock Loop)



Memory split ratio = 3 : 1
OF ARM

Data memory

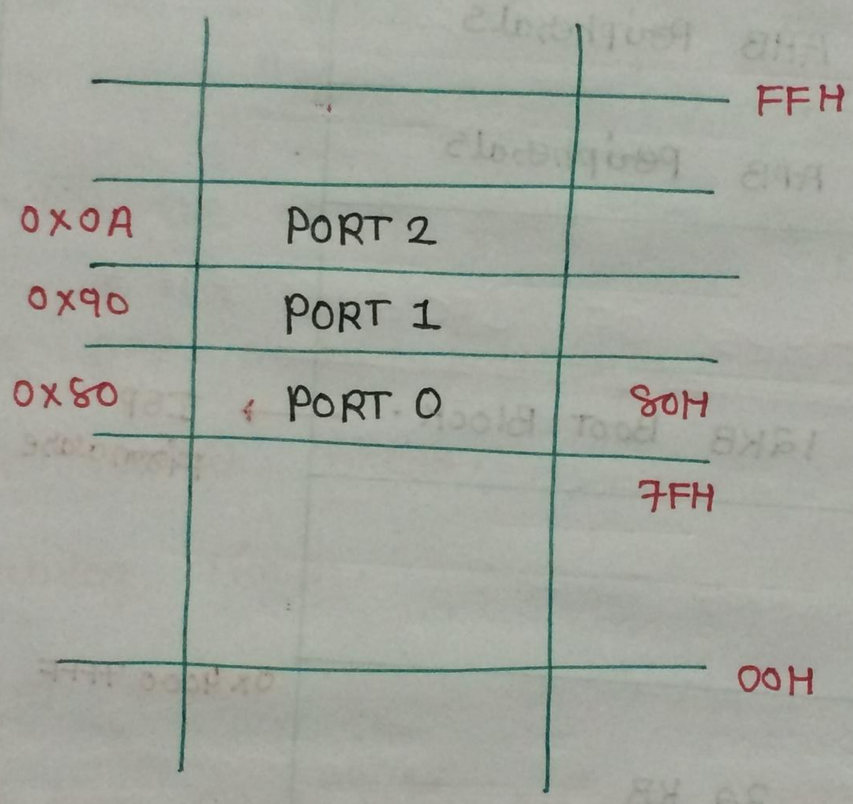
program memory

⇒ onward 1 GB RAM is start.

⇒ 32 KB use as a RAM.

⇒ 3.5 GB to 3.75 GB → APB peripheral mapped.

⇒ 3.75 GB to 4 GB → AHB peripheral mapped.




⇒ RAM contants Address - 0X4000000

Flash m/r Address - 0X0000000

⇒ after reset the immediately isp firmware is active.

⇒ The program is used to read the opcodes from serial cable and put it into a flash memory its called a ISP Firmware.

⇒ In ARM without startup code program is not compile.

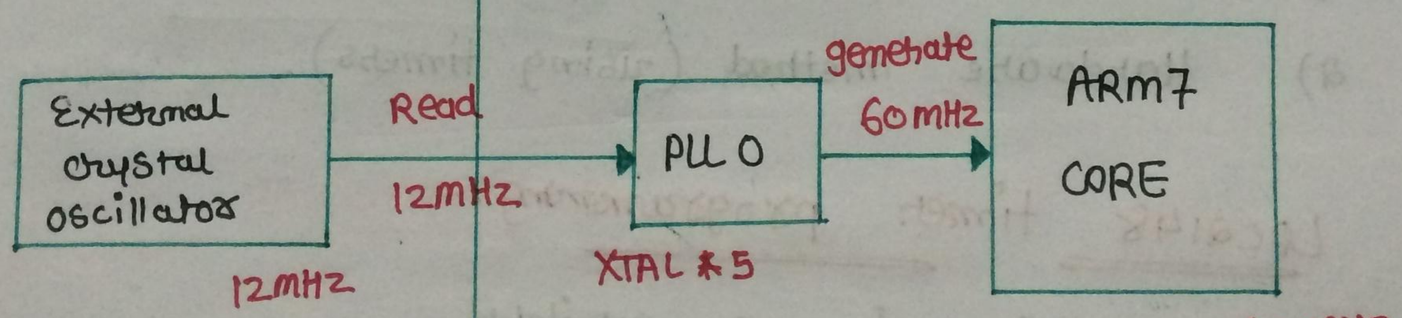
Startup code

| Yes ✓ | No |
|---|---|

## ⊛ Startup code of LPC2148 :-

1. Initialise Register's
   - VPB DIV
   - PLL Register's

   etc
   ⋮
2. Initialise CPU clock and peripheral clock.
3. Startup will configure & Enable PLL.
4. Set ARM7 processor to USER mode.
5. Call main() function.
6. Startup code will setup stack for each mode.
   → every mode stack address is different.



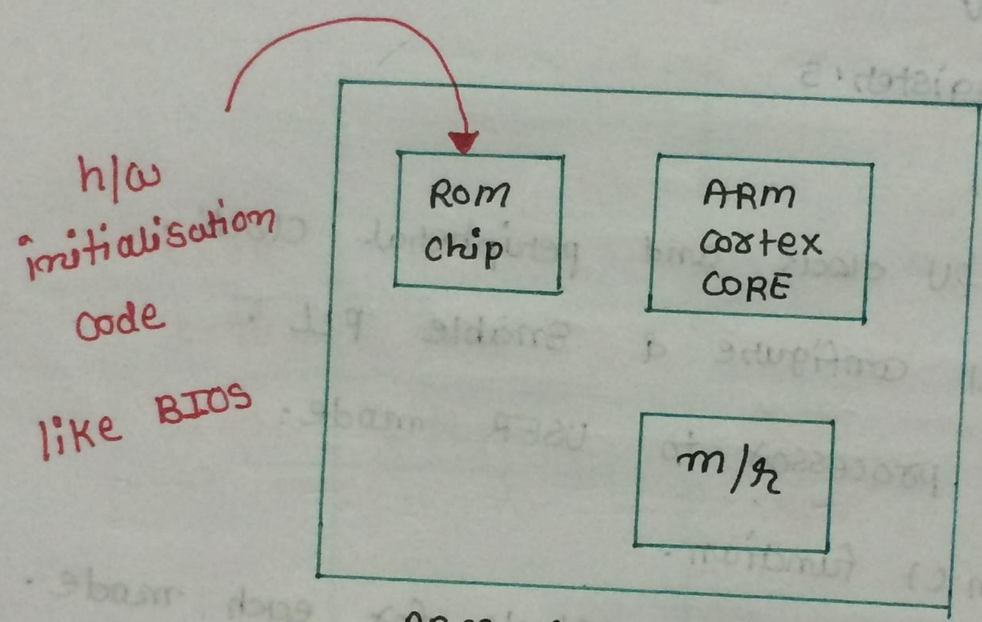

minimum multiplication value of PLL = 1

Maximum multiplication value of PLL = 6

two PLL peripherals available

- PLL0 (It is only generate processor clock)
- PLL1 (It is only used to generate USB Clock) → 48 MHz Frequency

→ USB 2.0 is used 48 MHz frequency so its is slower.




ARM Cortex SoC's.

## ⊛ Delay generation in LPC2148 :-

1) Software methode (using loops)
2) Hardware method (using timers).

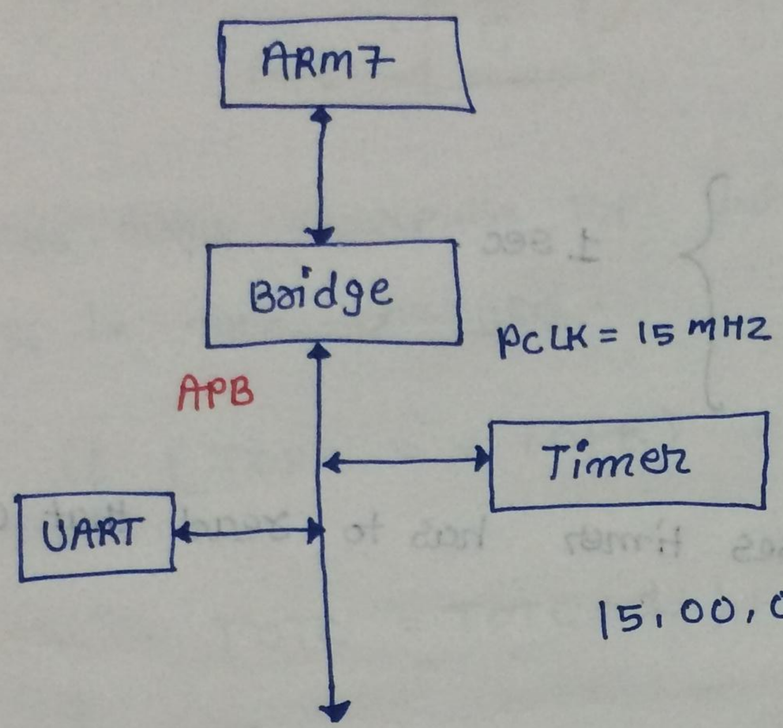
## ⊛ LPC2148 timer programming :-

Two 32 bit timer's available.

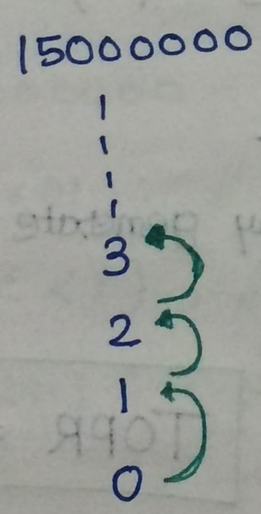
- timer 0
- timer 1

⇒ time slice is decide using timer only.

| |
|---|
| 8051 timer tick == 12 clock pulse. |
| 1 tick of LPC2148 timer == 1 clock pulse. |




15,00,00,00 clocks/sec

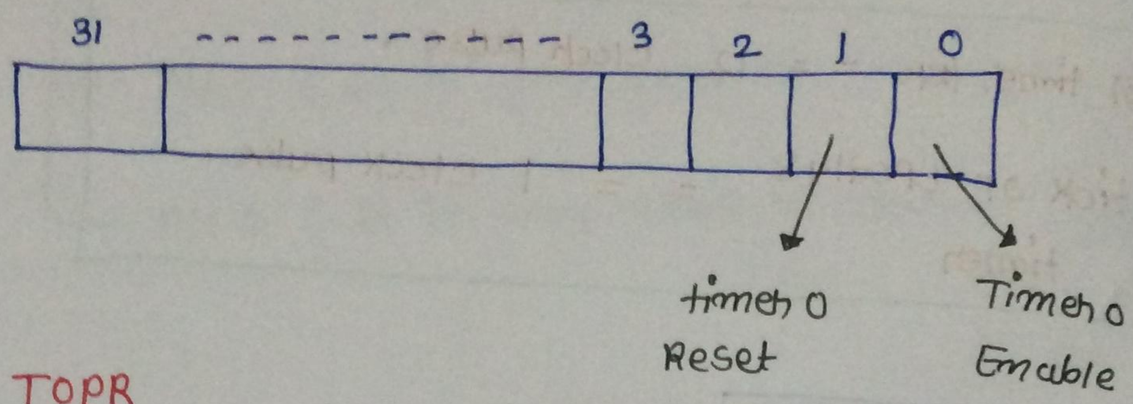



1 sec for read all the state

## Timer Registers :-

| Timer 0 | Timer 1 |
|---|---|
| T0TCR | T1TCR (timer control Reg). |
| T0PR | T1PR (timer prescalar reg). |
| T0TC | T1TC (timer counter) |
| T0PC | T1PC (timer prescalar counter reg.) |

① **TOTCR :-**




② **TOPR**

15000000

0

} 1 sec.

How many clock pulses timer has to read that count used in TOPR.

eg:- TOPR = 15000000 - 1 ;

if PCLK = = 15 MHZ

The 1 sec delay generate.

Example :-

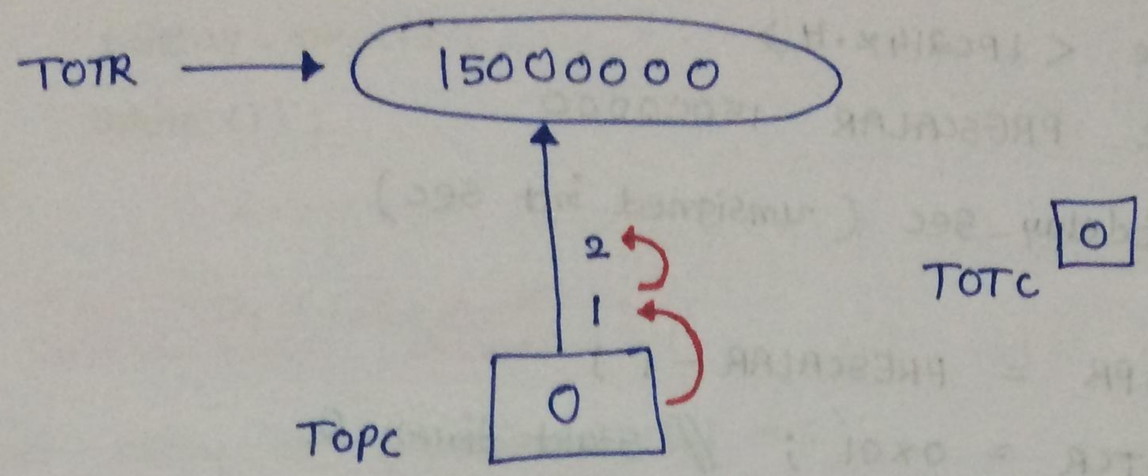
$$\frac{15000000}{1000}$$

$\boxed{\text{TOPR} = 15000}$

overflow after 1 mse

| Findout | prescalar Value |
|---|---|
| 10ms | 1500 |
| 100ms | 150 |
| 500ms | 300 |

## ③ TOTC




⇒ For every clockpulse TOPC value change. Value of TOPR is not changed.

```
if (TOPC == TOPR)
{
        TOTC = TOTC + 1 ;
}

TOPR = 15000000 - 1 ; //
TOTCR = 0x01 ; // start Timer
while (TOTC < 1) ; // monitor the OV Flag
TOTCR = (0x00) ; // STOP Timer
```

| headerFile is #include <LPC214X.H> |
|---|

in debugger.

## ⊛ 1 Sec delay Function :-

```
#include <LPC214X.H>
# define PRESCALAR 15000000
void tdelay_sec (unsigned int sec)
{
    T0PR = PRESCALAR - 1 ;
    T0TCR = 0x01 ; // Start timer 0
    T0TC = 0x00 ;
    while (T0TC < sec) ;
    T0TCR = 0x00 ; // Stop timer 0
}
main()
{
    tdelay_sec (1) ;
    while (1) ;
}
```

## ⊛ 1 ms delay Function :-

```
# include <LPC214X.H>
# define PRESCALAR 15000
Void tdelay_ms (unsigned int ms)
{
    T0PR = PRESCALAR - 1 ;
    T0TCR = 0x01 ; // Start timer 0
    T0TC = 0x00 ;
    while (T0TC < ms) ;
    T0TCR = 0x00 ; // STOP timer 0
}
```

```
main()
{
    tdelay_ms(1);
    while(1);
}
```

## Software method :-

```
Void delay_ms (unsigned int ms)
{
    unsigned int Y;
    for (  ; ms>0 ; ms--)
        for (Y = 12000 ; Y>0 ; Y--);
}
```

## ⊛ I/O programming :-

LPC2148 GPIO's pins :-

- PORT 0 (P0.0 to P0.31)
- PORT 1 (P1.16 to P1.31)

⇒ First 16 pin's of port 1 is use for Hardware debugging.

| |
|---|
| Don't use<br>P1.0 to P1.15 |

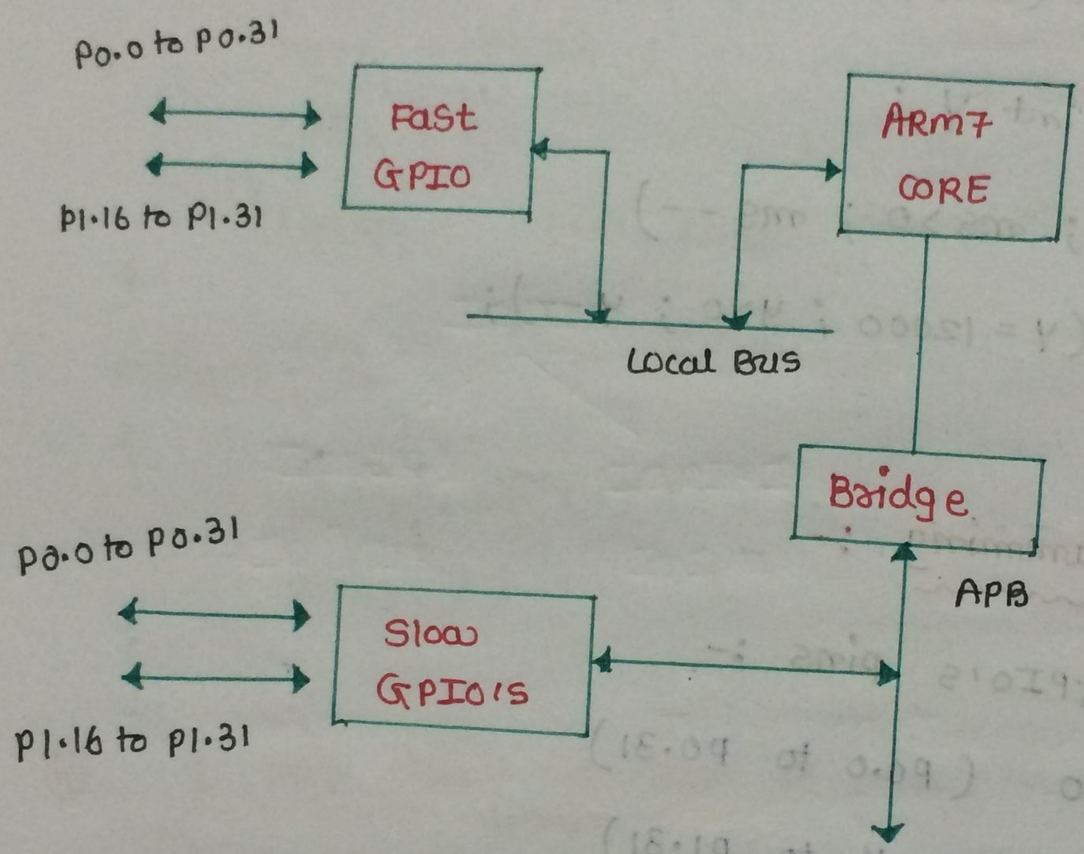
PORT 0 :- pins P0.24, P0.26, and P0.27 are not available.

## ⊛ Total 45 pins available for GPIO'S.

### Types of GPIO'S

1) Fast GPIO pins (60 MHz speed).

2) Slow GPIO pins (15 MHz speed).




### ⊛ Fast GPIO Register'S :-

1) SCS (System control & status Flag register)

2) FIODIR (Fast direction reg.)

3) FIOSET (Fast i/o pin set reg.)

4) FIOCLR (Fast i/o pin clear reg.)

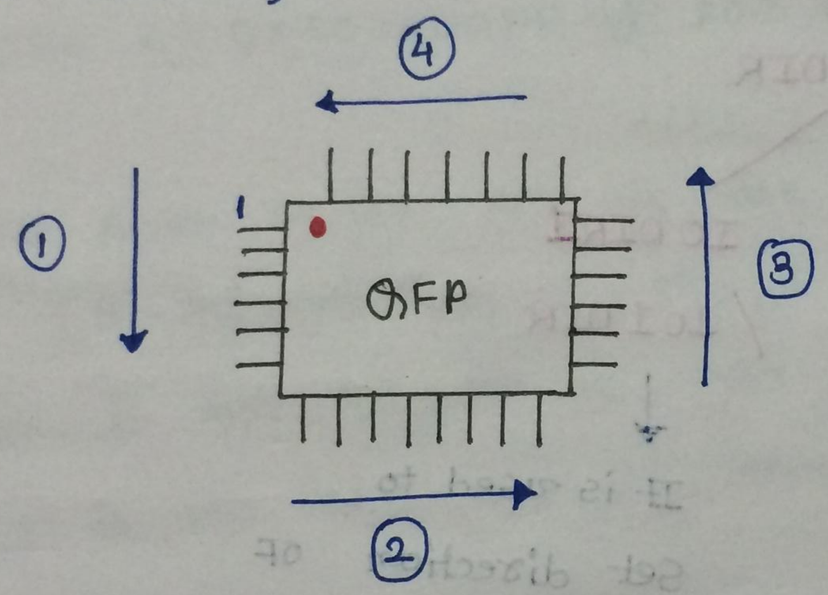
5) FIOPIN (Fast i/o pin reg. to read port pins).

## (*) Slow GPIO's registers :-

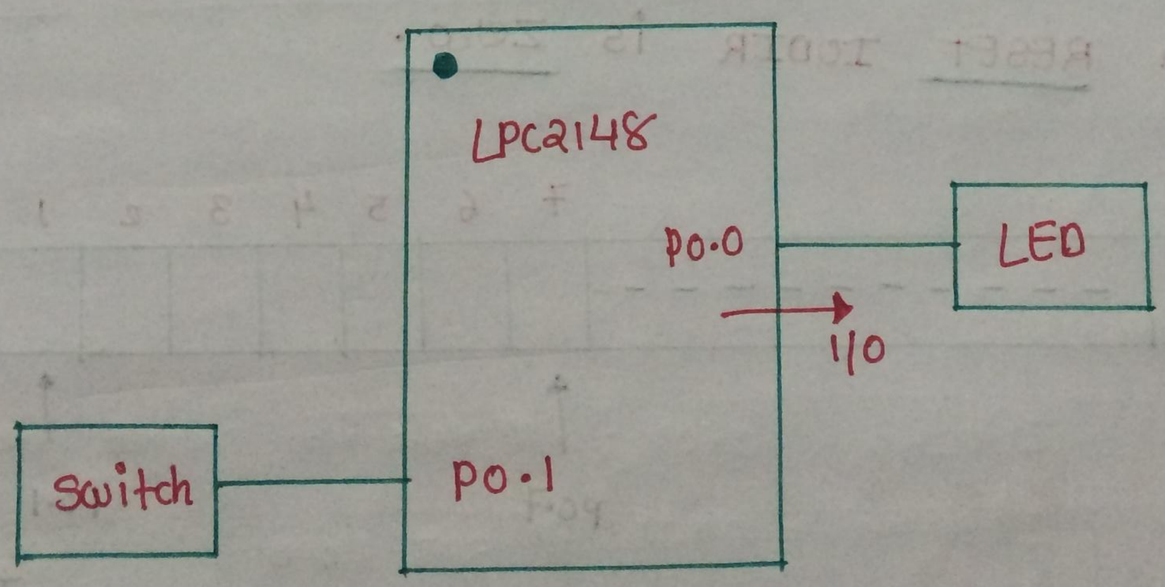
1) IODIR (direction reg.).
2) IOSET (i/o pin set reg.).
3) IOCLR (i/o pin clear reg.).
4) IOPIN (i/o pin reg. to read port pins).

## (1) IODIR :-

⇒ This register is set I/O pin direction (i.e output or input direction)




⇒ ARM is a Quad flat Package.



→ write only pin the set direction pin is 1.

```
if (P0.1 == 0)   // reading P0.1 status.
{
    Switch is pressed;
}
```

→ reading the switch status the direction is input direction.

| Output direction | input direction |
|---|---|
| LED, 7 segment, LCD | switch, Keypad. |

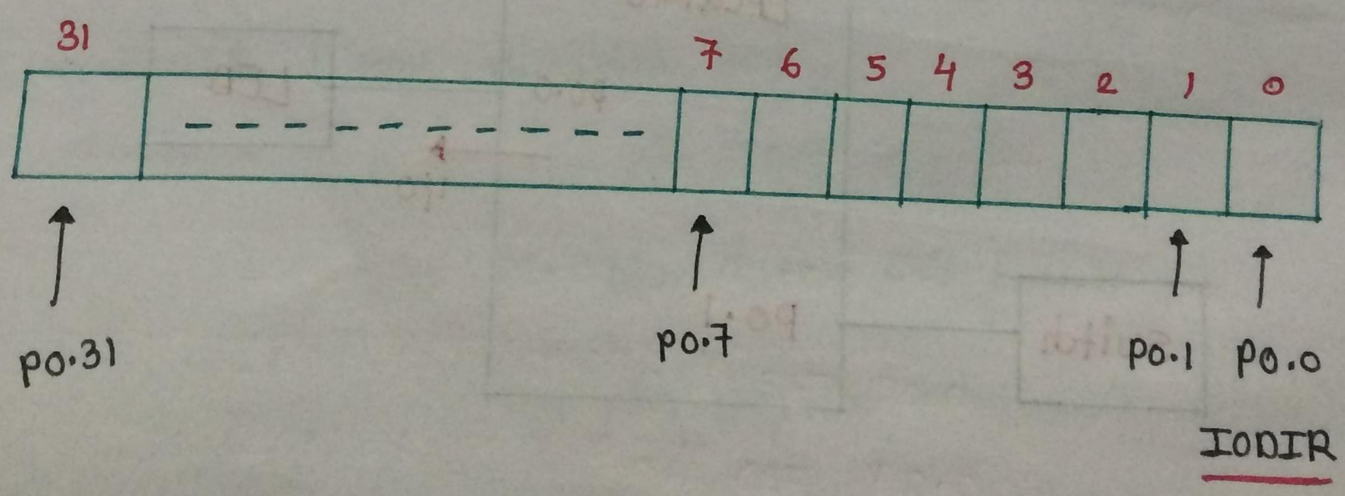
IODIR

IODIR0 / IO1DIR

IODIR0 → It used to set direction of PORT 0.

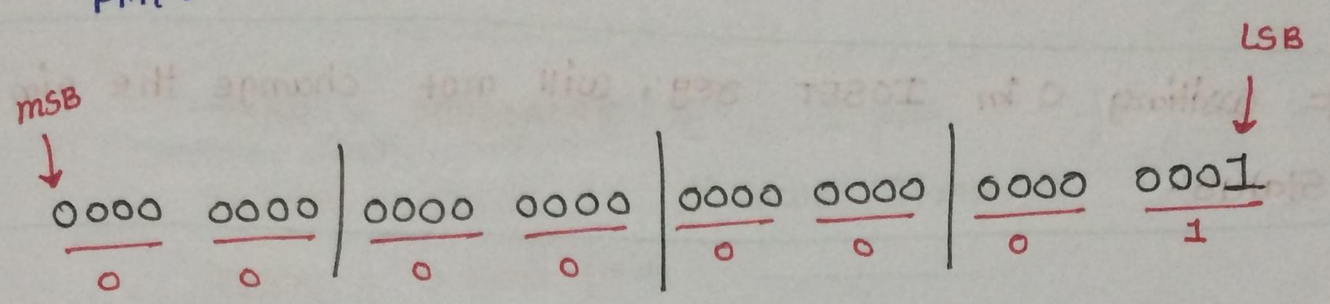
IO1DIR → It is used to set direction of PORT 1.

upon RESET IODIR is Zero.



⇒ IF 0 bit OF IODIR is 0 then P0.0 pin is used for a input pin.

⇒ IF 0 bit of IODIR is 1 then P0.0 as a output pin.




IODIR0 = 0X00000001 // P0.0 is output pin.

IODIR0 = 0X000000FF // P0.0 to P0.7 pin's are O/P pin's.

⇒ upon Reset all the port pin are high impedance state (Hi-Z) neither 0 nor 1.

⇒ IOSET is use for set the ARM pin.

⇒ IOCLR is use for clear the ARM pin.

## (2) IOSET

zero

1) IOSET0 / IO0SET

2) IOSET1 / IO1SET

one

upon RESET IOSET is zero.

| 31 | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | ---------- | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

⇒ IF IOSET0 is 1 means P0.0 bit is set.

IOSET0 = 0x00000001 ; // set P0.0 pin

IOSET0 = 0x0000000F ; // set P0.0 to P0.7 pins

> if writing 0 in IOSET reg. will not change the pin status.

## ③ IOCLR :-

zero

1) IOCLR0 / IO0CLR

2) IOCLR1 / IO1CLR

one

⇒ writing 1 to this reg. to change pin status to 0 writing 0 to this register will not change the pin status.

| 31 | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | ---------- | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

IOCLR0

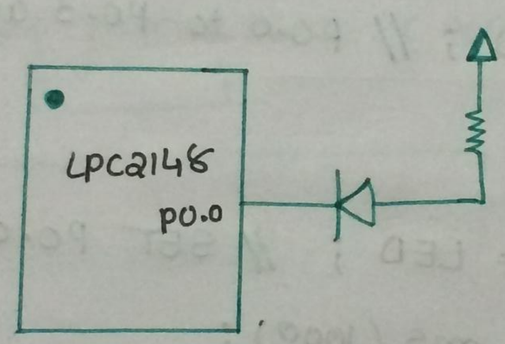
⇒ if IOCLR of bit0 is 1 than clear the P0.0 pin.

```
# include <LPC214X.H>
main()
{
        IODIR0 = 0x01 ; // P0.0 is output pin
        IOSET0 = 0x01 ; // SET P0.0
        IOCLR0 = 0x01 ; // CLR P0.0
        while(1);
}
```

## tdelay_ms header File :-

```
# define PRESCALAR 15000
Void tdelay_ms (unsigned int ms)
{
    T0PR = PRESCALAR - 1 ;
    T0TCR = 0x01 ;   // Start Timer 0.
    T0TC = 0x00 ;
    while( T0TC < ms) ;
    T0TCR = 0x00 ;   // Stop Timer 0.
}
```

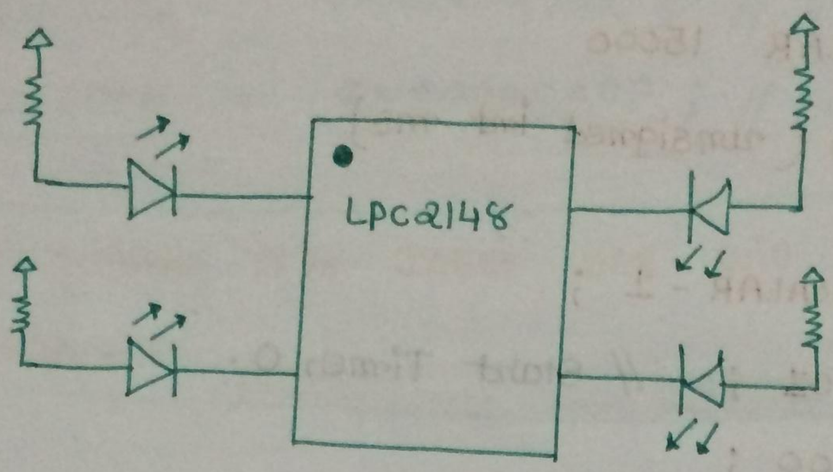
**Program 1** LED blinking ON P0.0 :-




```
#include <LPC2148.H>
# include " tdelay.h "
# define LED 0x01

main()
{
    IODIR0 = LED ;  // P0.0 is output pin.
    while(1)
    {
        IOSET0 = LED ;   // SET P0.0 , OFF LED
        tdelay_ms (1000) ;
        IOCLR0 = LED ;   // CLR P0.0 , ON LED
    }   }   tdelay_ms (1000) ;
```

Program 2

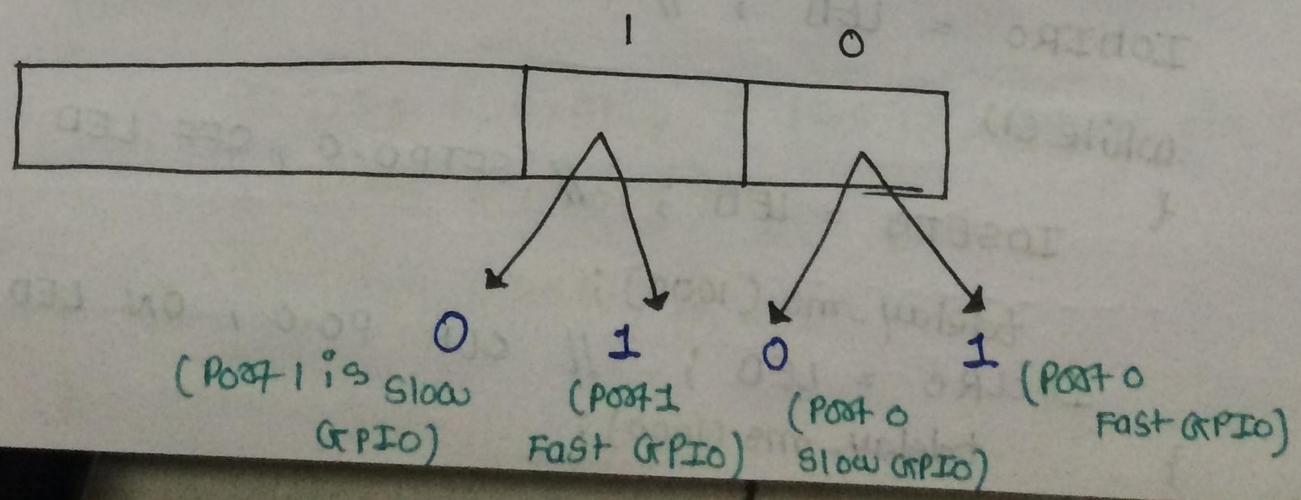
# Binking 4 LED .....




P0.0 and P0.1 is Tx and Rx so downloading time not connect it.

```
# include < LPC214X.H >
# include " tdelay.h "
# define LED 0XF0

main()
{
    IODIR0 = LED ; // P0.0 to P0.3 as output pins
    while(1)
    {
        IOSET0 = LED ; // SET P0.0 , OFF LED'S
            tdelay_ms (1000) ;
        IOCLR0 = LED ; // CLR P0.0 , ON LED'S
            tdelay_ms (1000) ;
    }
}
```

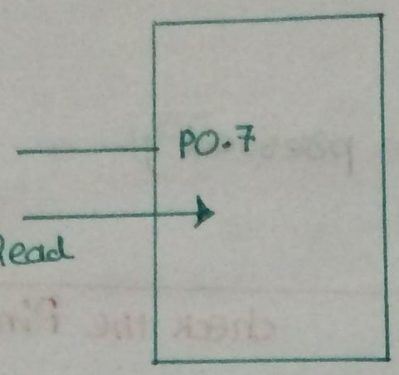
SCS



## ④ IOPIN :-

→ It is used to Read / write on ports.

→ IO PIN can not IOSET , IOCLR .




### I/O PIN Register's :-

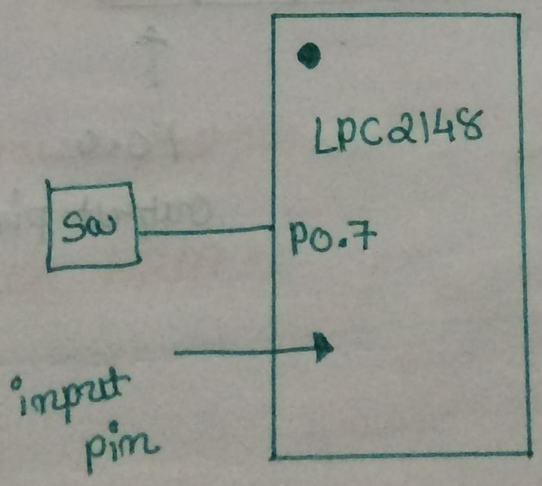
1) IOPIN0 / IO0PIN (PORT 0)

2) IOPIN1 / IO1PIN (PORT 1)

upon Reset it is high

| 31 | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

IOPIN0



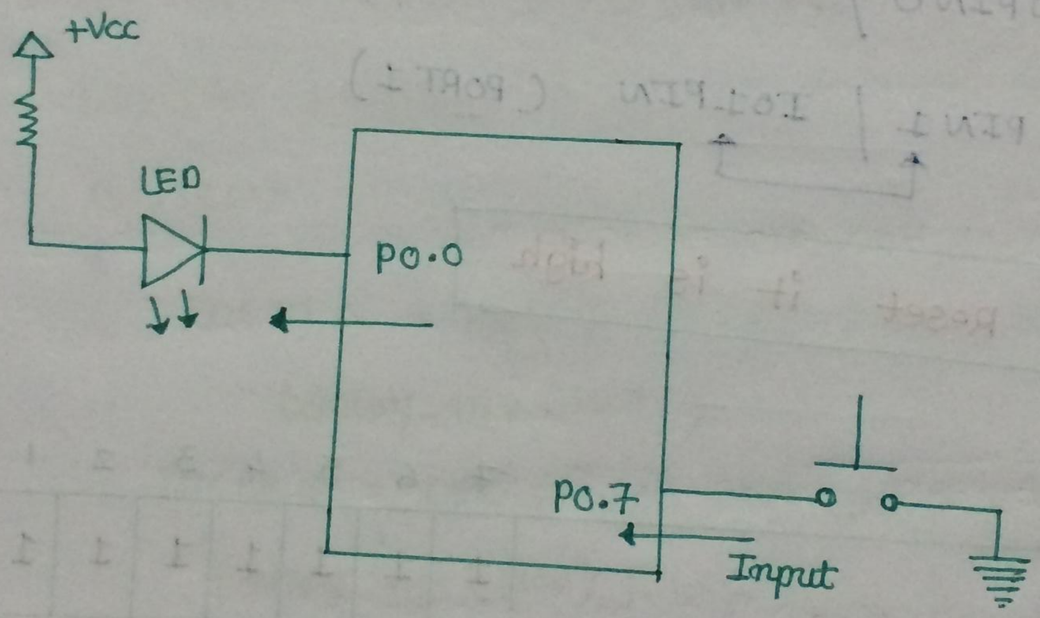

Press the switch

P0.7 = 0 .

```
if ((IOPIN0 & (1<<7)) == 0)
{
        switch is pressed;
}
else
{
        switch is not pressed;
}
```

check the Pin status

## -: Switch Interfacing :-




| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

P0.7 input pin

P0.0 Output pin

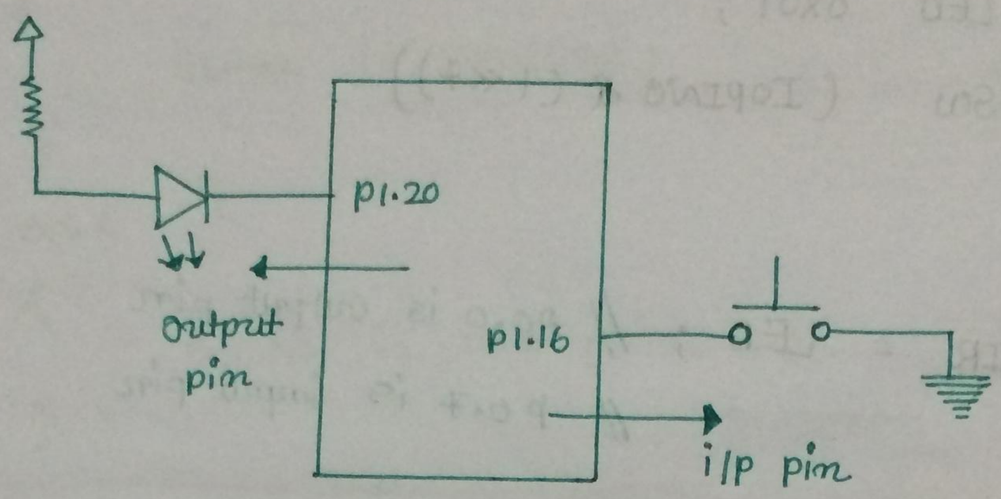
```c
#include <LPC214X.H>
#include "tdelay.h"
#define LED 0x01;
#define SW (IOPIN0 & (1<<7))

main()
{
    IODIR = LED; // P0.0 is output pin
                 // P0.7 is input pin
    while(1)
    {
        if (SW==0) // if SW is pressed
        {
            tdelay_ms(100); // avoid key bouncing
            IOCLR = LED; // LED ON
            while (SW==0); // waiting for SW Release.
        }
        else
        {
            IOSET0 = LED;
        }
    }
}
```

**Proteus :-**

LPC2138

External pull up Register is Mandatory for PIC / AVR OR ARM.

**program :-** LED Connected to P1.20 and Switch is connected to P1.16.




```
#include <LPC214X.H>
#include "tdelay.h"
#define LED (1<<20)
#define SW (IOPIN1 & (1<<16)) ;

main()
{
    IODIR1 = LED ; // P1.20 is o/p and P1.16 is i/p
    while(1)
    {
        if (SW==0) // if SW is pressed.
        {
            tdelay_ms (100) ; // avoid key bouncing
            IOCLR1 = LED  // LED ON
            while (SW==0) ; // waiting for Release
        }
        else // IF SW is not pressed
        {
            IOSET1 = LED ;
        }
    }
}
```

# *** Keypad program ***

Row's → output's

Col's → input's

R0 = P1.16

R1 = P1.17

R2 = P1.18

R3 = P1.19

```
IOCLR1 = IOCLR1 | (1<<16) | (1<<17) |
                (1<<18) | (1<<19)
```

C0 = P1.20

C1 = P1.21

C2 = P1.22

C3 = P1.23

```
IOSET1 = IOSET | (1<<20) | (1<<21) | (1<<22) | (1<<23)
```

**prog** 4x4 Keypad matrix using LPC2148.

Lcd always connected on Port 1.

## Using UART :-

⇒ Content of the file is send using UART.

⇒ File is not transfer using UART.

⇒ only using networking protocol we send file protocol is TFTP, FTP, SCP.

(use of the Network Cable).

### Advantages :-

1) UART code size is very less.
2) login to ARM based SOC's using UART.
3) It is used for Basic debugging for single Board computer.

⇒ Init is a part of the root file system (RFS).

/home
/etc
/media
/proc
/sys
/bin
} RFS

/user ← init present

⇒ Restart the computer than SBC is initialize the UART0.

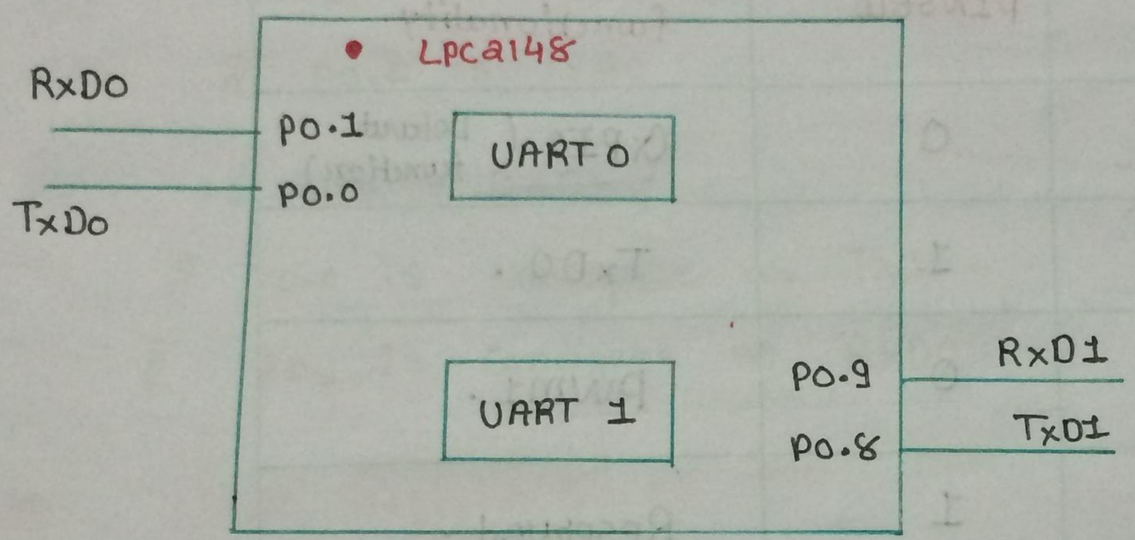
⇒ Network is up after starting the Kernel.

## ⊛ LPC2148 UART :-

TWO UART'S → UART 0 ( NULL modem )
→ UART 1 ( Full modem Support )

⇒ When 3 pin use of UART is call Null modem.

⇒ all the 9 pin are use for uart is call Full modem, it is use in telephone system.




⇒ ARM microcontroller pins are multi-Functionality pins.

⇒ ARM7 based microcontroller have maximum 4 Functionality.

## ⊛ Register :-

It is select for the special Functionality.

PINSEL ( Pin function select reg ) :-

1) PINSEL 0 ( Port 0 ) ( 32 bits ).

2) PINSEL1 ( Port 0 ) ( 32 bits ).

3) PINSEL2 ( Port 1 ) ( 32 bits ).

| |
|---|
| Port 1 → debug purpose pin. |

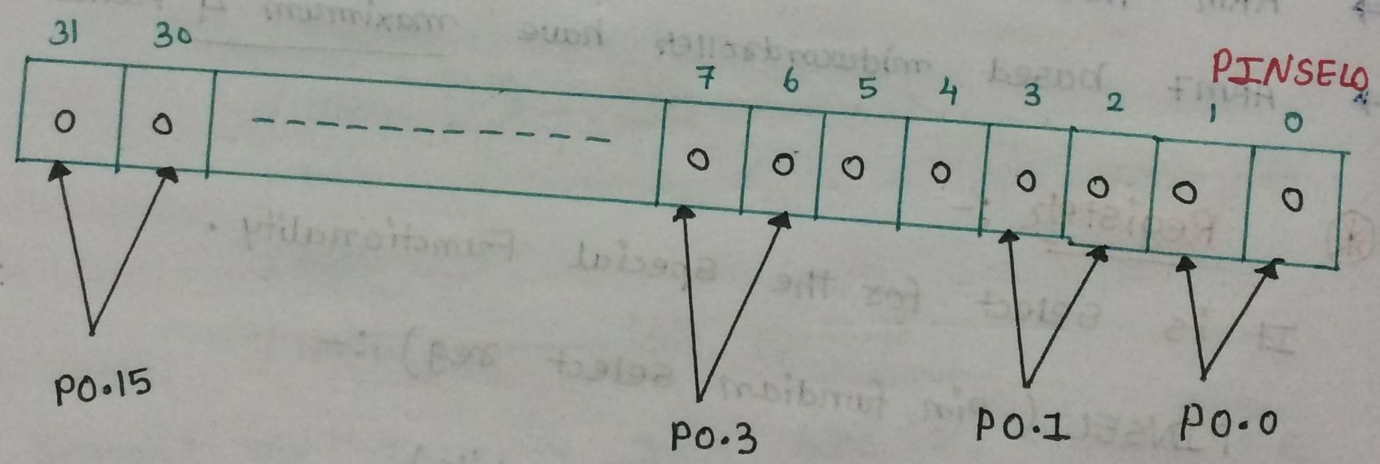
① **PINSEL0 :-**

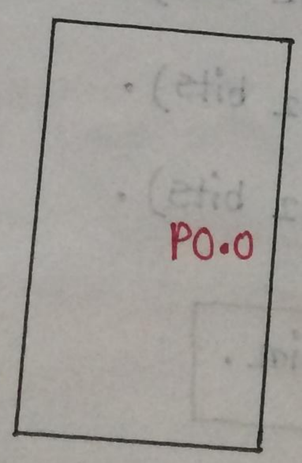
set special functionality of P0.0 to P0.15.

② **PINSEL1 :-**

set special functionality of P0.16 to P0.31.

| 1st bit of PINSEL0 | 0th bit of PINSEL0 | PIN P0.0 functionality. |
|---|---|---|
| 0 | 0 | GPIO (Defaut function) |
| 0 | 1 | TxD0. |
| 1 | 0 | PWM1. |
| 1 | 1 | Reserved. |



P0.0 GPIO / TxD0 / PWM1 / Reserved

↓

IOSET = 1 (PIN.0 = 1)

IOCLR = 0 (PIN.1 = 0)

| | | |
|---|---|---|
| 0 | 0 | Default Functionality |
| 0 | 1 | 1st special Functionality |
| 1 | 0 | 2nd special Functionality |
| 1 | 1 | 3rd special Functionality |

(1) PINSEL0 = 0x00000001 ;

// P0.0 is TxD0

(2) PINSEL0 = 0x05 ;

// P0.0 is configure as TxD0.

// P0.1 is configure as RxD0.

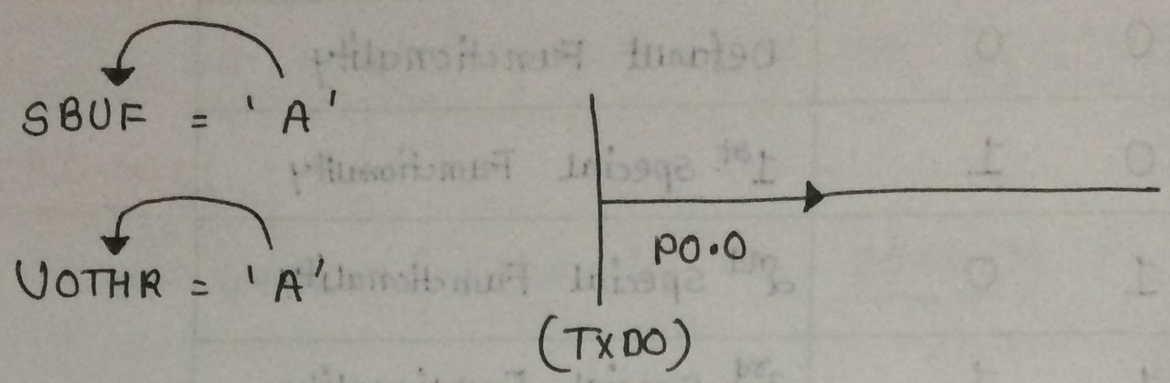
## (*) UART Register's :-

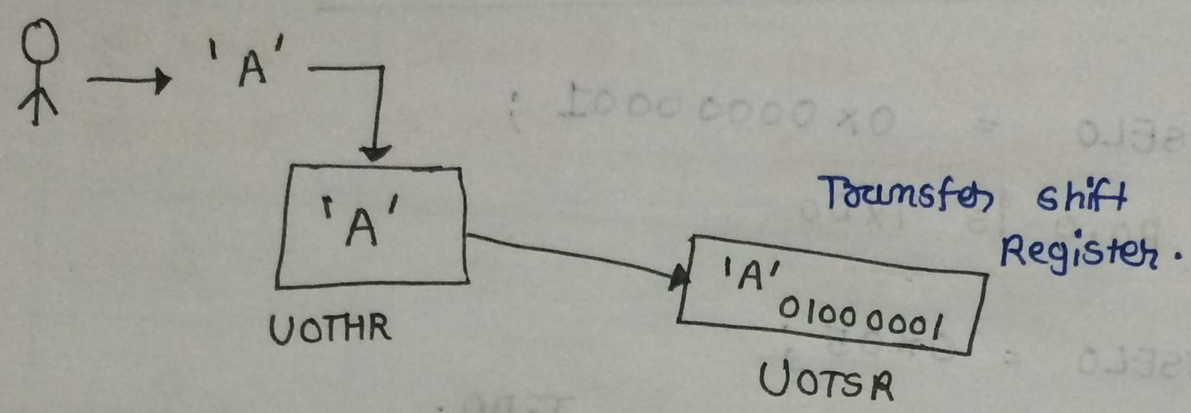
| UART0 | UART1 | Detail. |
|---|---|---|
| U0THR | U1THR | transmit holding Reg. |
| U0RB2 | U1RB2 | Receiver Buffer |
| U0DLL | U1DLL | divisor Latch LSB |
| U0DLM | U1DLM | divisor Latch MSB |
| U0LCR | U1LCR | line Control Reg. |
| U0LSR | U1LSR | line Status Reg. |
| U0FCR | U1FCR | FIFO Control Reg. |

Lab manual Page number - 148 LPC2148 manual

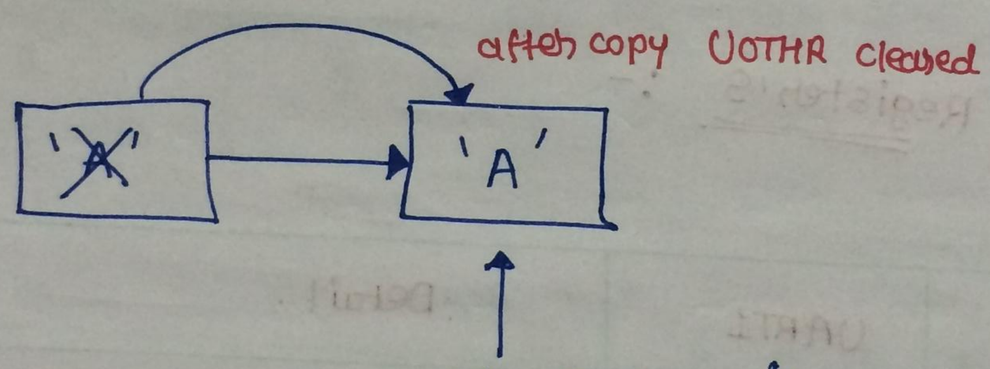
more Detail

## ① U0THR (UART Transmit holding Reg.) :-




⇒ The data is transfer from U0THR to U0TSR after that process data is cleared in to U0THR.




This Register is not access using program because it is internal for UART.

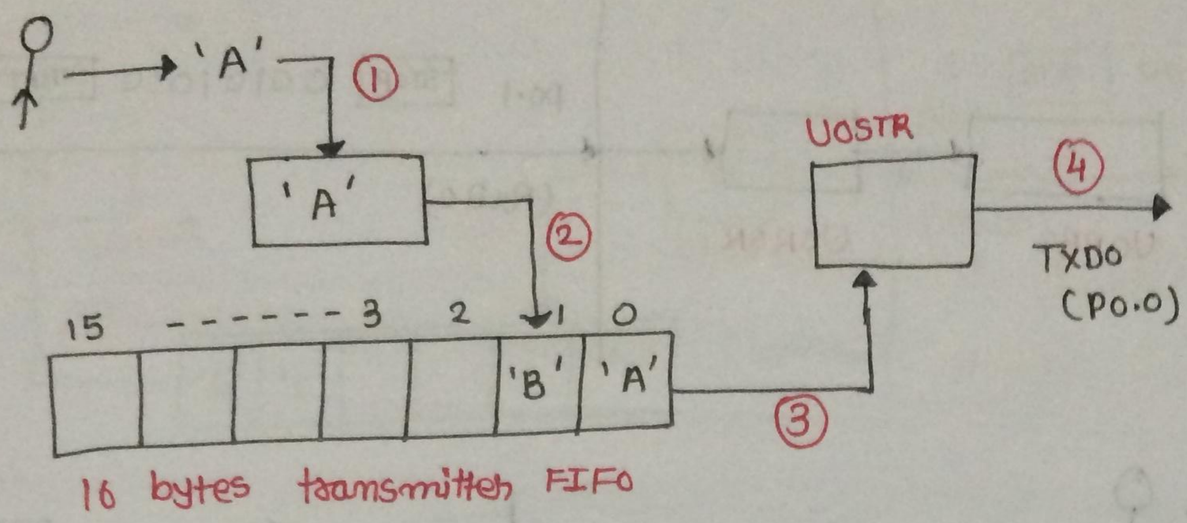
⇒ In LPC2148 16 bytes of RxTx FIFO is available.

⇒ FIFO is avoid the data corruption.

⇒ By default FIFO is disable.

⇒ To enable the FIFO use the U0FCR Register

(FIFO control Register).




⇒ we can't control the FIFO but we get the FIFO status.

- FIFO is Full.
- FIFO is empty.

```
SBUF = 'A';
SBUF = 'B';
```

→ data is corrupt B is only copy in SBUF.

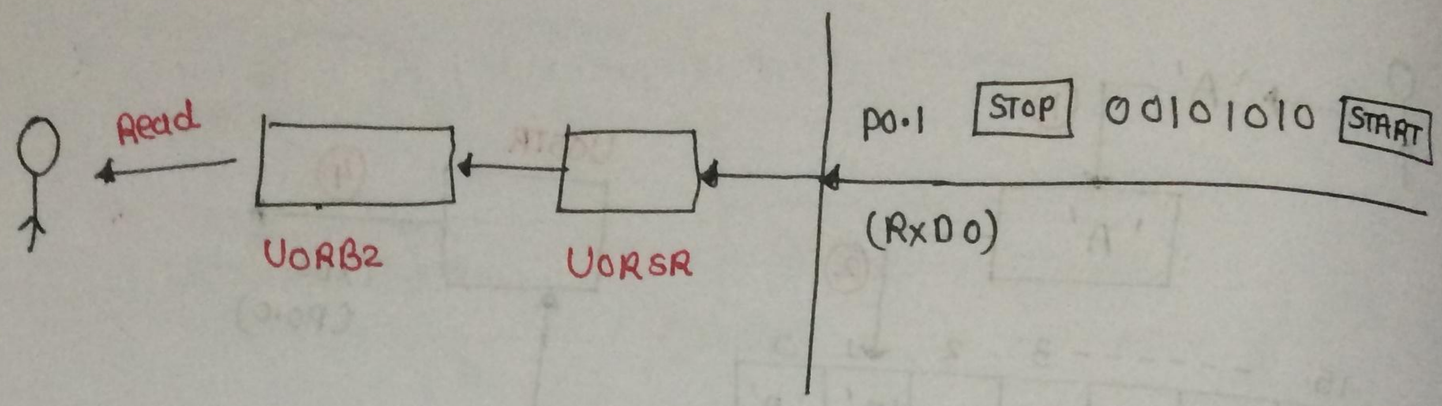
⇒ UART we not wait the fill FIFO.

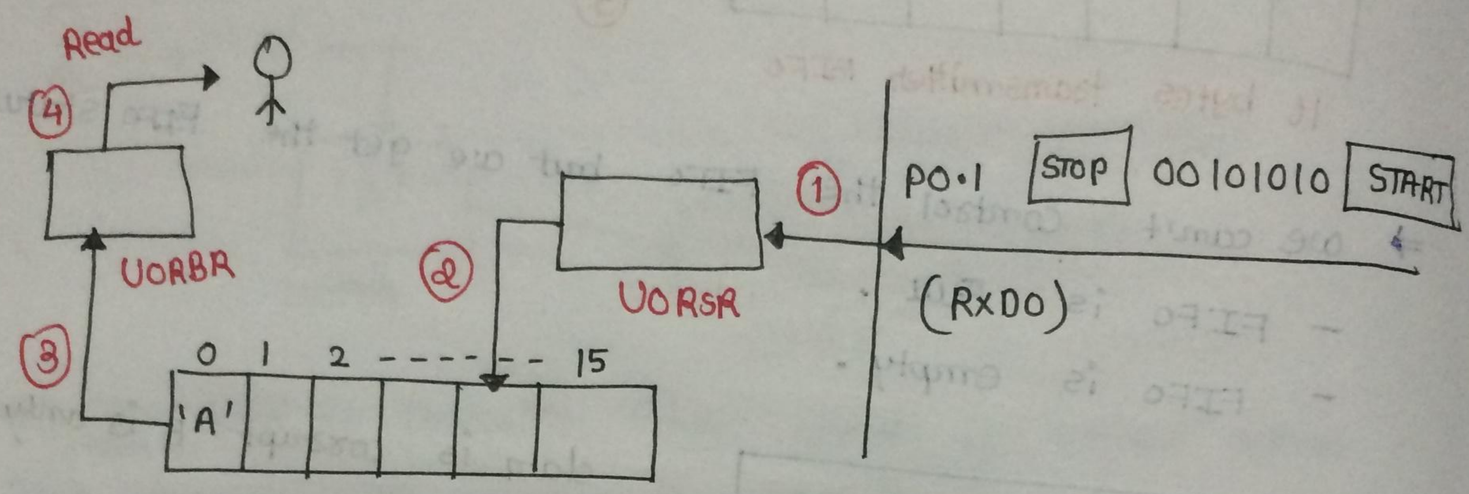
⇒ In LPC2148 'A' is store in FIFO, again immeidiate B is store in FIFO.

⇒ maximum 16 bytes data store into FIFO.

② **U0RBR** ( UART0 Receiver buffer)

```
Var = SBUF   // read SBUF

Var = U0RBR
```




→ after Reading RB2 Register then RBR is empty.

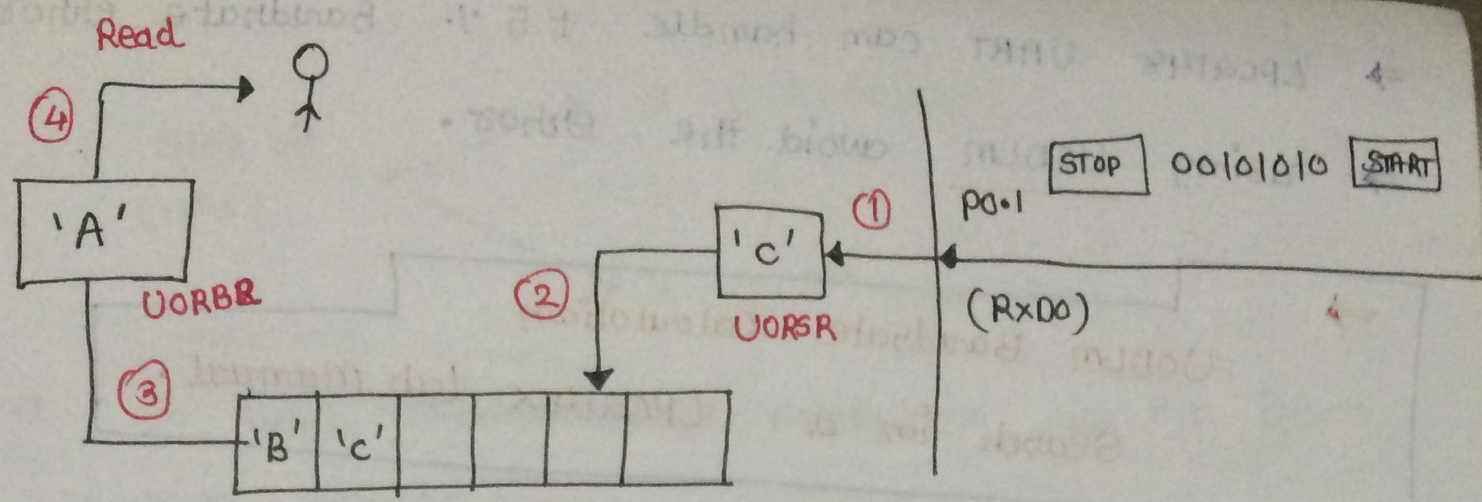
→ The device transmit next char 'B' it is copy in UORSR and after store in FIFO then Store in UORBR.

→ The RBR is not Empty that time data Store in FIFO next to next memory and FIFO is Full than new Character assembler in RSR Then UART 0 is Generate the FIFO error.

("overrun error")

Three type OF Register :-

1) Data.
2) Status.
3) control.




## 3 UODLL & UODLM :-

UODLL → Devicer Latch LSB

UODLM → Devicer Latch MSB

→ using this two Registers set Baudrate.

## * Baudrate Generation Formula :-

Working Frequency of UART

PCLK = 15 MHz.

$$\boxed{UODLL = \frac{PCLK}{16 * Baudrate}}$$

$$\therefore UODLL = \frac{15000000}{16 * 9600}$$

$$= \frac{15000000}{153000}$$

$$\therefore \boxed{UODLL = 97.65625}$$

→ only integer Value.

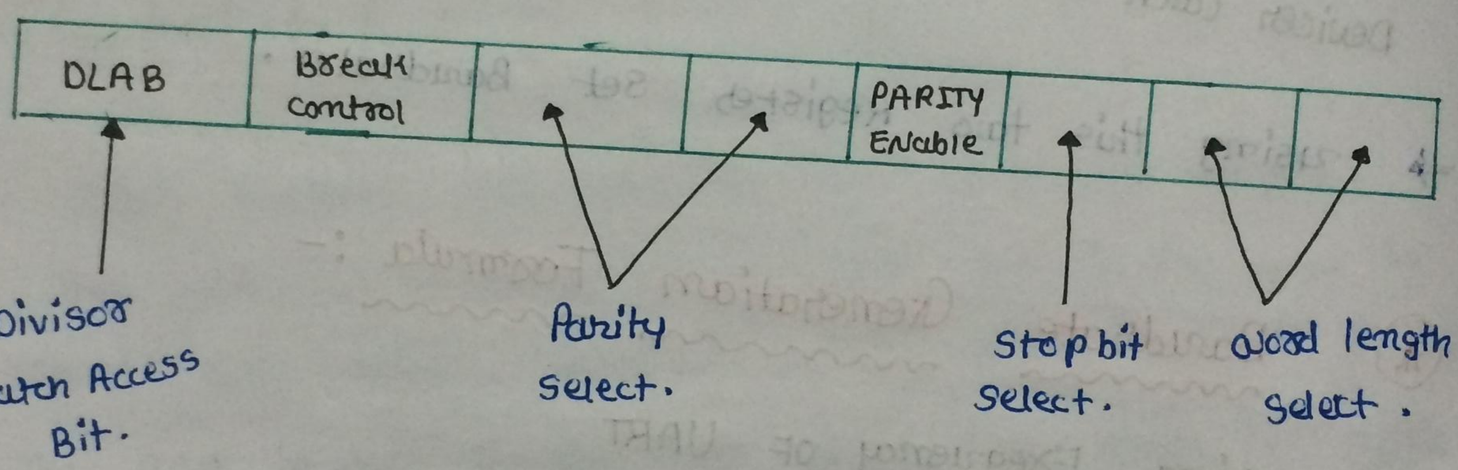
⇒ LPC2148 UART can handle ± 5 % Baudrate error.

⇒ using U0DLM avoid the error.

⇒ U0DLM Baudrate Calculation search in a LPC214x lab manual.

④ U0LCR (Page :- 155) lab manual LPC214X.

⇒ UART0 Line control Register.

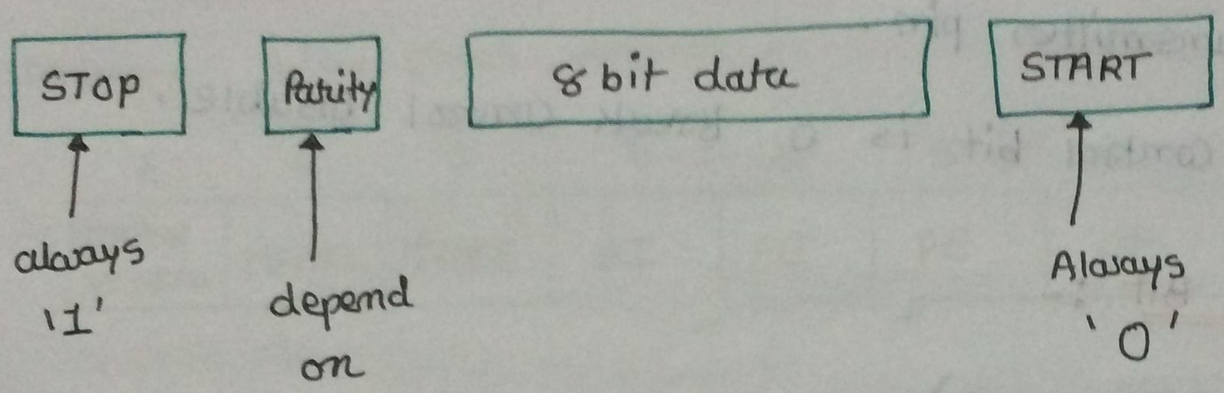



① word length select :-

| word length | | Mode |
|---|---|---|
| 0 | 0 | 5 bit char's |
| 0 | 1 | 6 bit char's |
| 1 | 0 | 7 bit char's |
| 1 | 1 | 8 bit char's |

STOP bit (2nd) → 0 (1 Stopbit)
STOP bit (2nd) → 1 (2nd stop bits)

⇒ In UART communication parity is use for Error checking purpose.




## ⊛ Even Parity :-

⇒ If no. of 1's are even in data bits the parity Bit = 1.

## ⊛ Odd parity :-

⇒ If no. of 1's are odd in data bits the parity Bits = 1.

## ⊛ Parity select :-

| 5th | 4th | Parity |
|---|---|---|
| 0 | 0 | Odd. |
| 0 | 0 | even. |
| 1 | 0 | Forced to "1". |
| 1 | 1 | Forced to "0". |

⇒ Force to "1" means 1 is transmitted without any even or odd parity.

⇒ Force to "0" means 0 is transfer without any even or odd parity.

⇒ if 3rd bit is 0 then no effect of 4th and 5th bits.

⇒ Break control bit is 1 only continuously zero transmitted from transmitter pin.

⇒ Break control bit is 0 Break control Disable.

## ⊛ DLAB Bit :-

```
if (DLAB == 1)
{
    ① set Baudrate
        UODLL & UODLM
    ⇒ DLAB == 1 than only change the UODLL & UODLM).
    ⇒ DLAB == 0 than not modify the (UODLL & UODLM)
    ② can't transmit
      can't receive
}

if (DLAB == 0)
{
    ① we can't change the baudrate.
    ② UART0 can transmit can Receive data.
}
```

## program UART Initialization :-

```
U0LCR = 0x83 ;    // DLAB = 1
U0DLL = 97 ;      } 9600 bps
U0DLM = 0 ;
U0LCR = 0x03 ;    // DLAB = 0
```

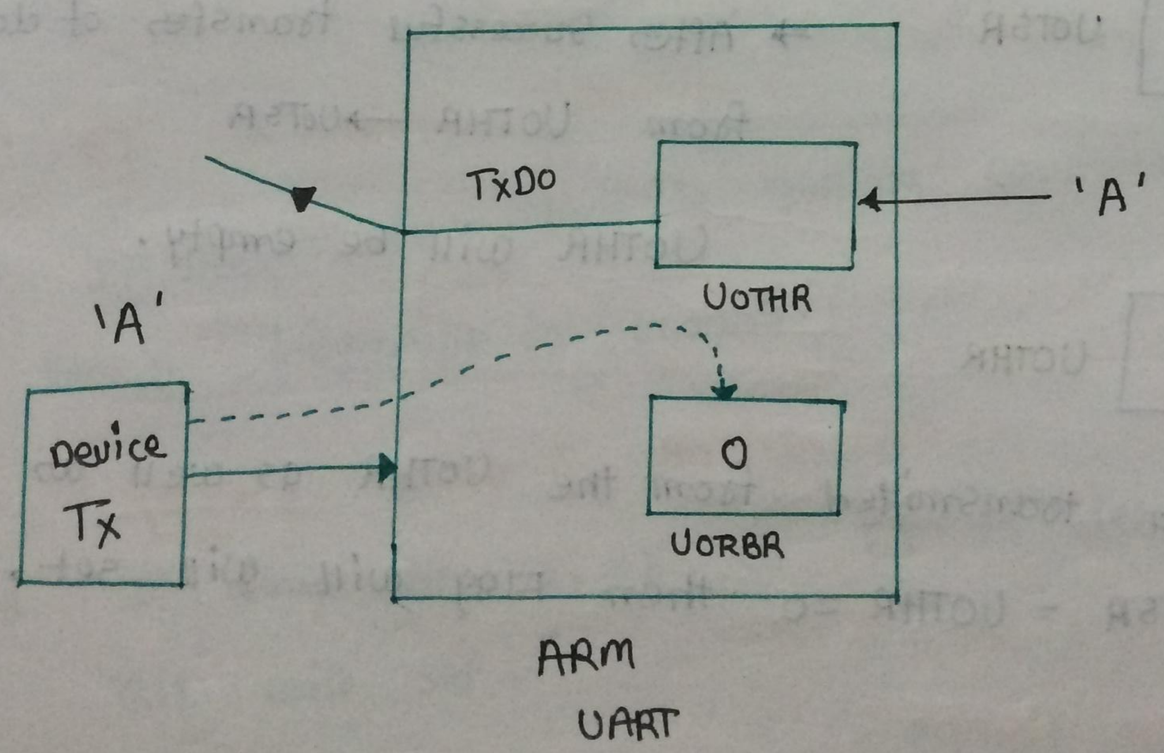
## 5 U0LSR :-

UART0 Line status Register.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ERROR IN FIFO | TEMT | THRE * | BI | FE | PE | OE | RDR * |

Receive DATA Ready (RDR)

- 0 (U0RBR is empty) no data Receive
- 1 (U0RBR contains Valid data) Data Received into U0RBR

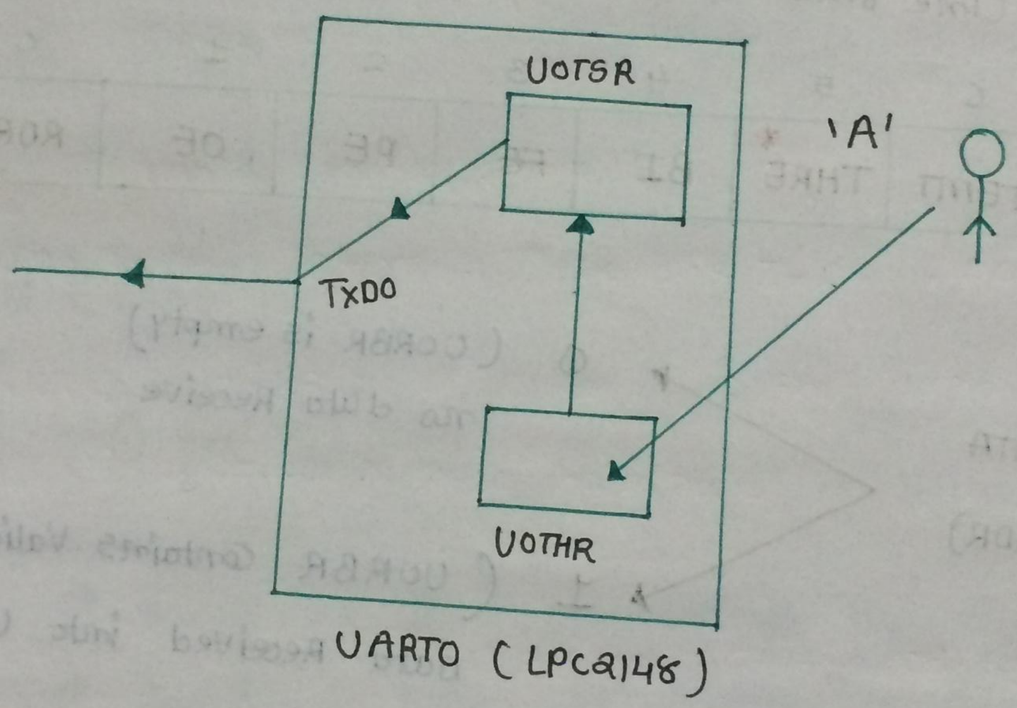


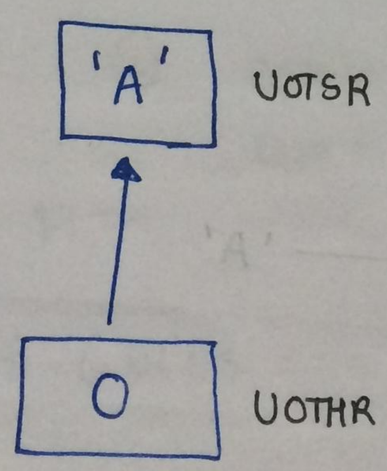
→ After Reading U0RBR Register RBR flag will clear automatically.

② **THRE :-**

Transmit Holding Register empty → 0 (U0THR contains Valid data)

Transmit Holding Register empty → 1 (U0THR is empty)




⇒ After successful transfer of data from U0THR → U0TSR

U0THR will be empty.

⇒ If data transmitted from the U0THR as well as from U0TSR = U0THR = 0 then Flag will will set.

TEMT → 1 (UOTSR = UOTHR = 0)

TEMT → 0 [while ((UOTSR || UOTHR) != 0);]

* **OE = Over Run Error :-**

⇒ If FIFO is full, and if new character is assembled into UORSR, then OE occurs.

* **PE = Parity Enable :-**

⇒ If parity is enable than Data bits is corrupted then PE occurs.

* **FE = Framing Error :-**

⇒ If stop bit received as a 0 then framing error occurs.

FE → 1 (Framing error occurs).

FE → 0 (no error).

* **BI = Break Interrupt :-**

⇒ If the receiver line is getting continuously zero than break interrupt is occurs.

* **RXFE :- Error in Receiver FIFO :-**

⇒ If any OE, PE, FE, or BI is detected then this Bit will set.

→ If all data bits is receive as a zero than break interrupt will generate.

ie:- BI became 1 at runtime.

## ⊛ LPC2148 uart programming :-

" uart0.h "

```
#define THRE (1<<5)
Void uart0_init()
{
    // uart0 initialization
    PINSEL0 = 0x05 ; // P0.0 is TxD0 , P0.1 is RxD0
    U0LCR = 0x83 ; // DLAM =1 ; 8bit word length
    U0DLL = 97 ; // 9600 baud rate
    U0DLM = 0 ;
    U0LCR = 0x03 ; // DLAB = 0 , 8bit word length
}
Void uart0_tx (unsigned char data)
{
    U0THR = data ;
    while ((U0LSR & (1<<5)) == 0);
                  THRE
}
unsigned char uart0_rx ()
{
    while ((U0LSR & 0x01) == 0);
    return U0RBR ;
}
```

```
Void uarto_string ( char *s)
{
    while (*s)
        uarto_tx (*s++);
}
```

**program:-1** Transmit one char.

```
#include <LPC214X.H>
#include "uarto.h"
main()
{
    uarto_init() ;
    while(1)
        uarto_tx ('A');
}
```

**program 2** :- Loopback program.

```
#include <LPC214X.H>
#include "uarto.h"
main()
{
    uarto_init() ;
    // loopback test code
    while(1)
        uarto_tx (uarto_rx());
}
```

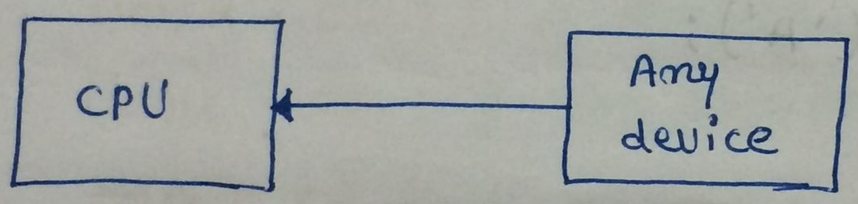
Program 3 transmit a string.

```
#include <LPC214X.H>
# include "uart0.h"
main()
{
    uart0_init();
    uart0_string ("SAVAN");
}
```

Vector Interrupt – "Same ISR"

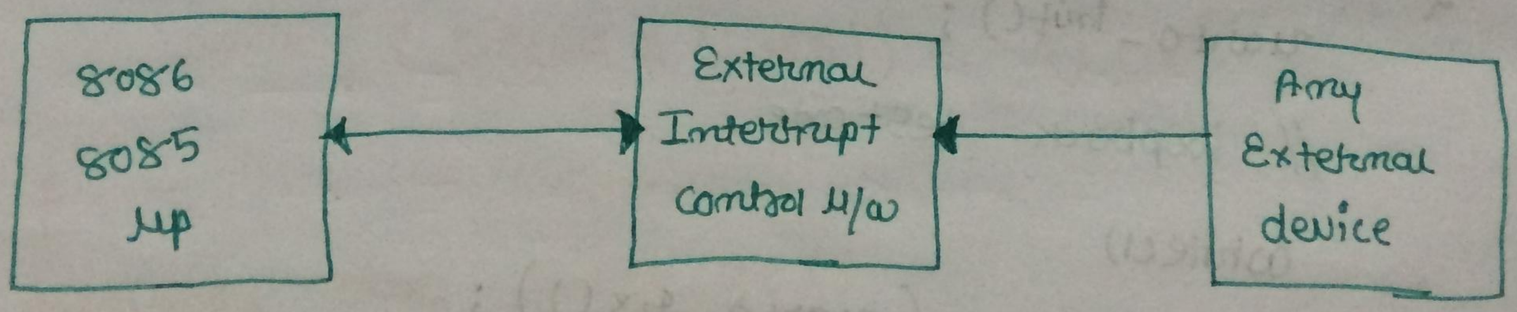
Non Vector Interrupt – "Common ISR"

⇒ every Vector Interrupt has own different ISR address.




⇒ In LPC2148 Interrupt can handled by Vectored Interrupt Controller.

⇒ In 8086, 8085 µp Interrupt handling we need external Interrupt Control Hardware.



⇒ In this case no internal Interrupt control H/w is present hence external interrupt control H/w is present.

⇒ seprate peripheral trying directly try to interrupt the ARM µp then pin no. will increase, execution speed issues, priority issues.

⇒ Instand on H/w control device
i.e VIC will do all these things.

⇒ If µp checks for priority, then the interrupt latency will increase.

⇒ VIC will run parallel with ARM µp at 60 MHZ.

## ⊛ Types of Interrupt :-

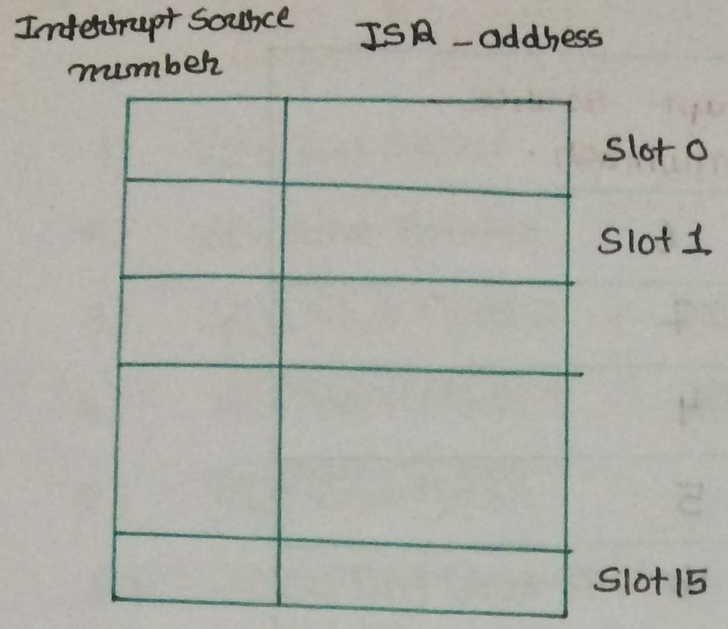
1) RESET
2) Software Interrupt (using SWI arm Instruction).
3) Hardware Interrupt (FIQ, IRQ)

## ⊛ LPC2148 Hardware Interrupt :-

⇒ Interrupt classification based on Interrupt priority.

1) Faster Interrupt Request (FIQ)
2) Vectored Interrupt Request (IRQ)
3) Non Vectored Interrupt Request (NVIRQ)

1) FIQ (High priority).
2) IRQ (middle priority).
3) NVIRQ (Low priority).




⇒ every peripheral of ARM can interrupt the processor

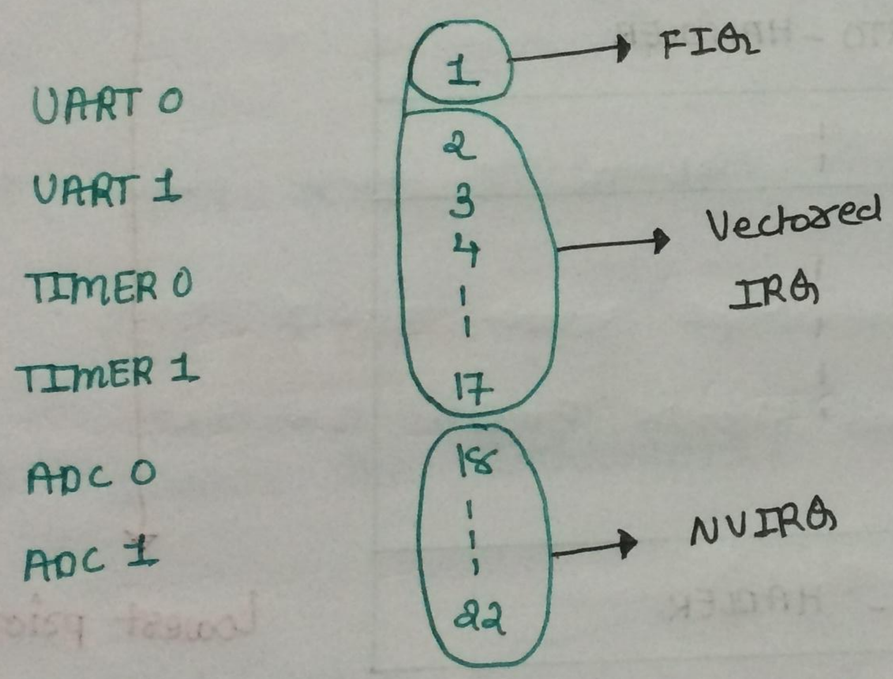
⇒ In LPC2148 total 22 Interrupt Source code are available

⇒ out of 22 Interrupt,

1 source is use as FIQ.

(FIQ is like UART0, UART1, Timer0, Timer1, ADC0, ADC1)

22 (total Interrupt)
− 1 (FIQ)
− 16 (VIRQ)
5




⇒ every Interrupt has a unique interrupt source number

Page 67

| Interrupt | Interrupt source number. |
|---|---|
| UART0 | 6 |
| UART1 | 7 |
| TIMER 0 | 4 |
| TIMER 1 | 5 |
| ADC0 | 18 |
| ADC1 | 21 |

| | Interrupt source number | ISR - Address |
|---|---|---|
| Slot 0 | 4 | TIMER0 - HANDLER |
| Slot 1 | 6 | UART0 - HANDLER |
| | ⋮ | ⋮ |
| | ⋮ | ⋮ |
| Slot 15 | 18 | ADC0 - HADLER |

Highest priority

Lowest priority.

VIC IRQ slots.

⇒ IRQ Interrupt Execution is same as 8051 µc.

⇒ IRQ Interrupt can't stop the execution of ISR of Running Interrupt.

⇒ Execution of IRQ's can't be preempted by IRQ's.

# VIC Registers :-

1) VIC Int Select
2) VIC Int Enable
3) VIC Vect Cntl 0 - VIC Vect Cntl 15
4) VIC Vect Addr 0 - VIC Vect ~~Cntl~~ Addr 15
5) VIC Vect Addr
6) VIC Def Vect Addr

## ① VIC Int Select (VIC Interrupt Select Register)

⇒ select IRQ or FIQ category.

VICIntselect → 1 (FIQ)
VICIntselect → 0 (IRQ)

⇒ upon reset VICIntselect register will be zero.

## ② VIC Int Enable :-

⇒ Enable the Interrupts.

VICInt Enable → 1 [Enable]
VICInt Enable → 0 [Disable]

## ③ VICVectCntl0 - VICVectCntl15 :-

⇒ These registers are used to assign a particular interrupt source to a particular slot.

UART0 → 6

TIMER0 → 4

EINT0 → 14

VICVect Cntl 0 = 6 ;

VIC Vect Cntl 15 = 14 ;

| Int-source | ISP address | |
|---|---|---|
| 6 | | Slot0 (Enable) |
| | | Slot1 |
| | | Slot2 |
| | | ⋮ |
| 15 | | Slot15 (Enable) |

⇒ After assigning Interrupt source no. of slot programmer must and should Enable the slot.

(*) How to enable the slot?

VICVectCntl 0 = 6

5th bit of VICvectCntl 0-15 is "Slot Enable bit".

VICVectCntl 0 = 6 | (1 << 5)

Enable slot → 0

```
VICVectCntl15 = 14 ; (1<<5) ;
```

Enable slot 15

## ④ VICVectAddr0 - VICVectAddr15 :-

→ For Vectored IRQ's, these Registers store the addr of the function, that must be called when an Interrupt occurs.

```
Void UART0_HANDLER (Void)
{
    ≡≡≡
}
Void EINT0_HANDLER (Void)
{
    ≡≡≡
}
```

```
VICVectAddr0 = (unsigned int) UART0_HANDLER ;
```

```
VICVectAddr15 = (unsigned int) EINT0_HANDLER ;
```

## ⑤ VIC Vect Addr :-

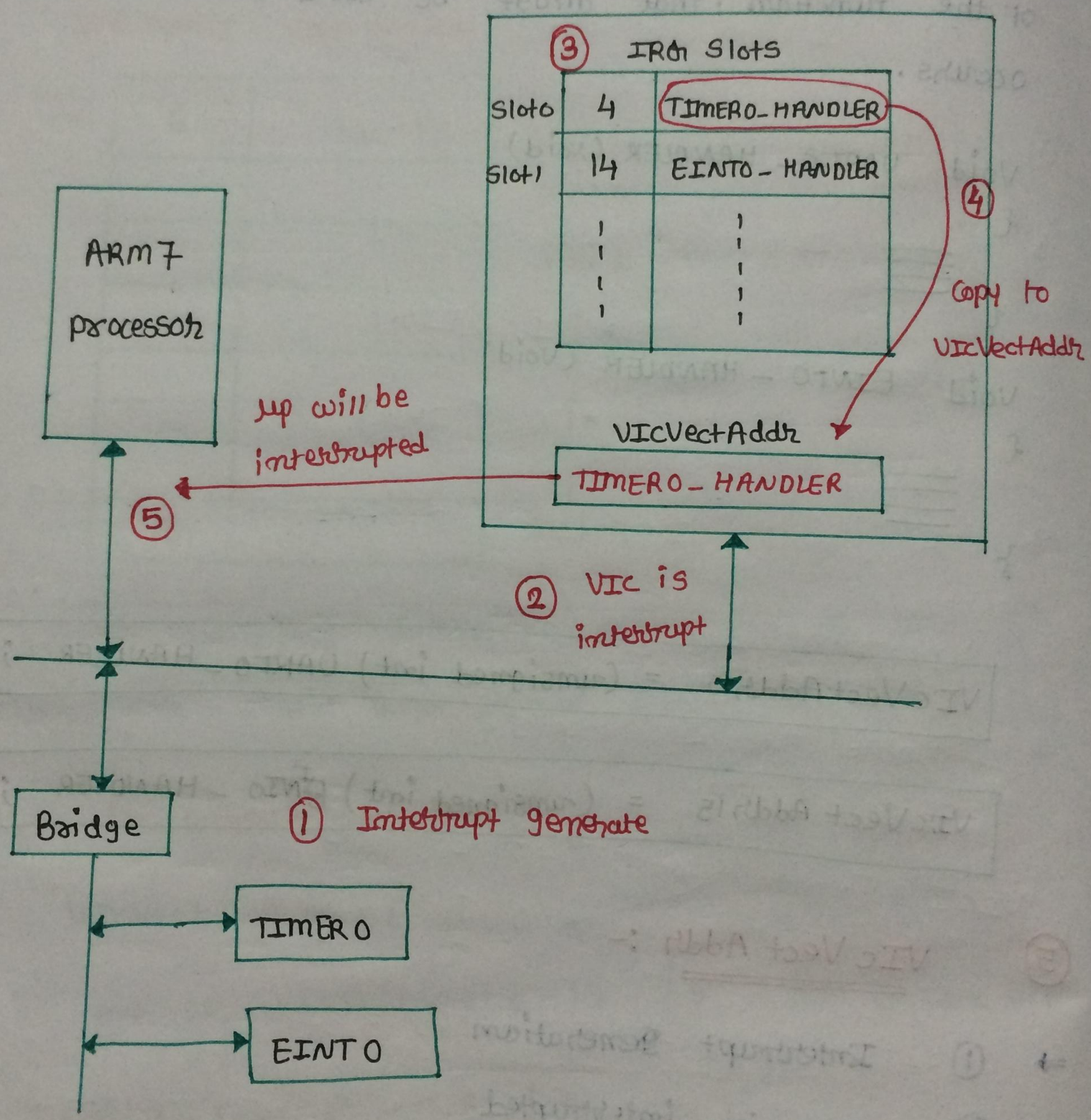
→ ① Interrupt Generation

② VIC is interrupted

③ VIC IRQ slot table is looked up.

④ The corresponding ISR - addr to the Interrupt generated will be copy to VIC VectAddr register.

⑤ Now µp will be interrupted to execute the interrupt ISR by µp picking up ISR address from VICVectAddr register present In VIC.



⇒ dummy write to VICVectAddr register will clears VICVectAddr register.

```
VICVectAddr = 0x00 ;   // dummy write
```

## ⊛ Non Vectored IRQs :-

⇒ Share common ISR.

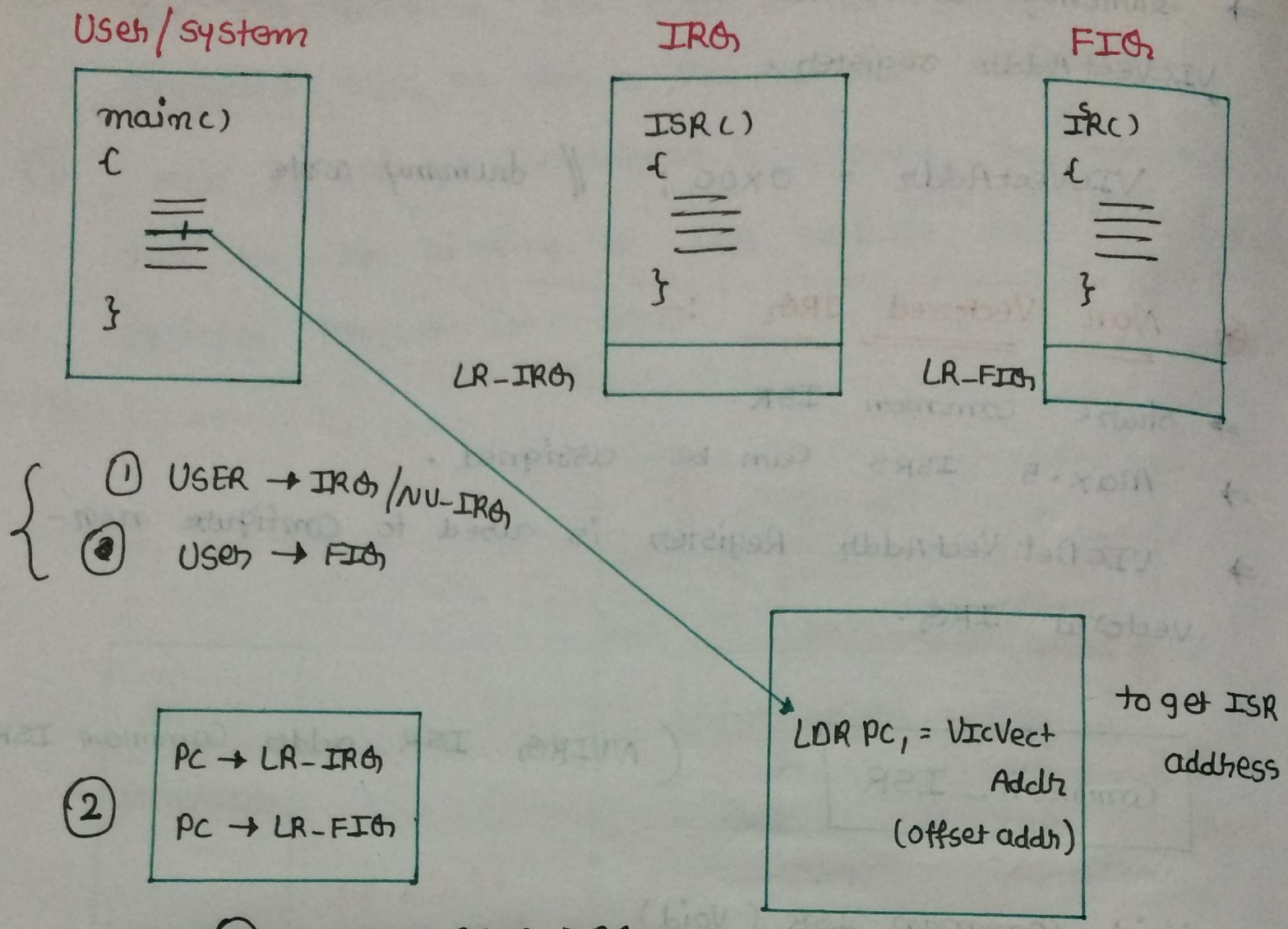
⇒ Max. 5 ISRs can be assigned.

⇒ VIC Def VectAddr Register is used to configure non-vectored IRQs.

$\boxed{\text{Common_ISR}}$ (NVIRQs ISR addr Common ISR).

```
Void Common_ISR (Void)
{
    if (pwm_Interrupt_flag == 1)
    {
        ========
    }
    if (UART1_Interrupt_flag == 1)
    {
        ========
    }

    VICVectAddr = 0x00 ;   // Imp.
}
```

Register is not cleared than process is not comeout from the ISR.




③ CPSR → SPSR_IRQ

CPSR → SPSR_FIQ

① USER → IRQ / NV_IRQ

USER → FIQ

② PC → LR_IRQ

PC → LR_FIQ

③ CPSR → SPSR_IRQ

CPSR → SPSR_FIQ

④ LDR PC, = VIC VectAddr

⑤ PC will point to ISR of IRQ or FIQ.

(6) LR_IRQ → PC
LR_FIQ → PC

(7) SPSR_IRQ → CPSR
SPSR_FIQ → CPSR

(8) FIQ / IRQ → user mode

(9) Remaining instruction in the main() will execute.

## ⊛ LPC2148 IRQ programming :-

Registers to be initialized

VICVectCntl0 - 15

VICVectAddr0 - 15

VICIntSelect

≡

VICIntEnable

## ⊛ How to define the ISR (IRQ / FIQ / NVIRQ) :-

```
void ISR_Name(void) __irq
{
    ≡
}
```

## ⊛ LPC2148 External Interrupt Handling :-

Total 4 external interrupt sources available.

EINT0 - EINT3

## ⊛ External Interrupt Registers :-

1) EXTMODE (to use to select level / edge sensitive)
2) EXTPOLAR (to use to select Falling edge / Rising Edge)
3) EXTINT (use to clear the interrupt Flags in ISR).

### ① EXTMODE :-

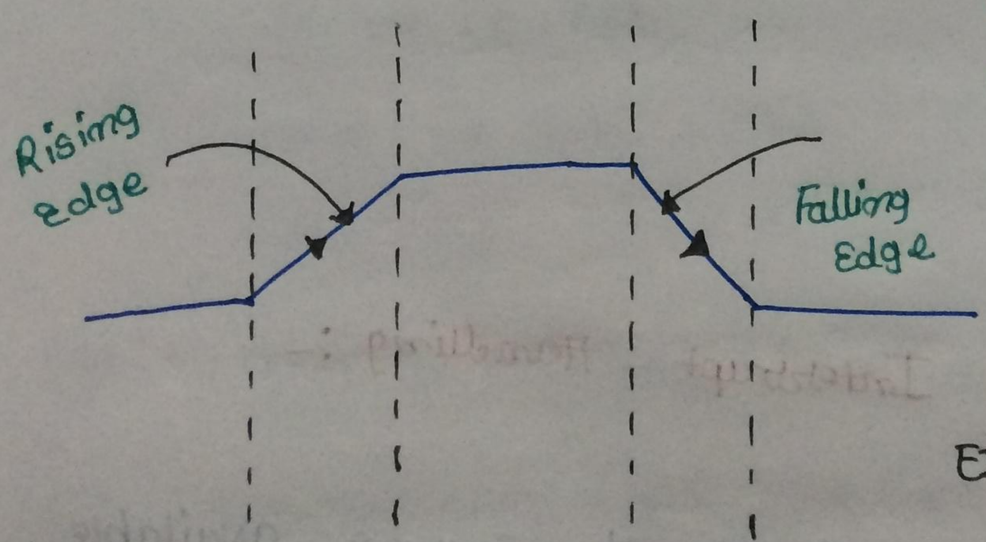
| 31 | | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|
| | ------ | EINT3 | EINT2 | EINT1 | EINT0 |

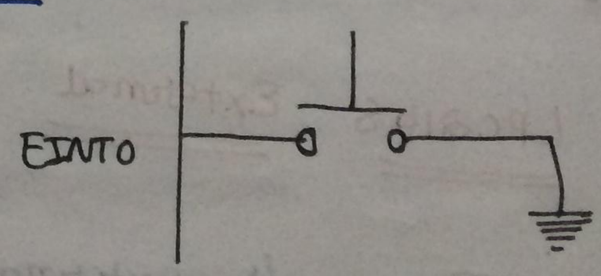
EINT1 → 0 : level Sensitive
EINT1 → 1 : Edge sensitive

### ② EXTPOLAR :-

| 31 | | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|
| | ------ | EINT3 | EINT2 | EINT1 | EINT0 |

EINT0 → 0 : (EINT0 is Falling Edge Sensitive)
EINT0 → 1 : (EINT0 is Rising Edge Sensitive)



⇒ upon switch press

pin status will change from 1 → 0

i.e on falling Edge, EINT0 will be fired.

and whole scenario vice versa.

⇒ In 8051 upon external interrupt generation IEx int will set and when entering ISR of that external interrupt, this IEx bit will be cleared automatically but in LPC 2148 it is not so.

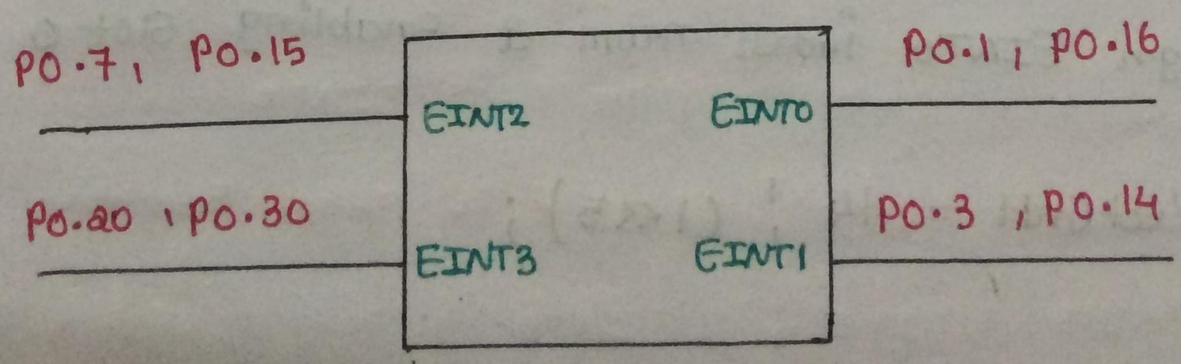
③ EXTINT :-

| EINT3 | EINT2 | EINT1 | EINT0 |
|---|---|---|---|

0 → no effect

1 → clear external interrupt flag.

```
void EINT0_ISR() __IRQ
{
    ≡≡≡≡
    EXTINT = 0x01;
    VICVectAddr = 0x00;
}
```



## In debug mode :-

RawIN will be set, as the program control enters into now

when, we do EXTINT = 0x01;

then only this

RawINT will change to 0.

↓

interrupt flag bit in debug mode.

**Program 1** Demo program for External Interrupt 0.

```
#include <LPC214X.H>
unsigned int counter = 0;
Void EINT0_HANDLER (Void) __irq
{
    Counter ++;
    EXTINT = 0x01; // clear EINT0 Flag
    VICVectAddr = 0x00; // Finish ISR
}
Void config_VIC ()
{
    VICIntSelect = 0x00; // All types are IRQ's

/* assign EINT0 inter num & enabling slot 0 */

VICVectCntl = 14 | (1<<5);
```

```
/* assigning EINT0 - ISR address to slot 0 */

    VICVectAddr = (unsigned int) EINT0 - HANDLER ;

    VICIntEnable = (1<<14) ; // enable.

}

Void config_EINT ()
{
    PINSEL1 = 0x01 ; // P0.16 is EINT0

    EXTMODE = 0x01 ; // EINT0 is edge Sensitive

    EXTPOLAR = 0x01 ; // EINT0 is Rising edge sensitive

}

main()
{
    unsigned int Count = 0 ;

    Config_EIN() ;

    Config_VIC() ;

    // default task in main.

    while(1)
    {
        Count++
    }

}
```

## * LPC2148 FIQ Handling :-

In startup code

// default :-

FIQ_Addr DCD FIQ_Handler

// New modify :-

FIQ_Addr DCD FIQ_HANDLER

IMPORT FIQ_HANDLER

Program 2 Demo program OF FIQ :-

```
#include <LPC214X.H>
unsigned int Counter = 0 ;
Void FIQ_HANDLER (Void) -- isq
{
    Counter ++ ;
    EXTINT = 0x01 ; // clear EINT0 Flag
    VICVectAddr = 0x00 ; // Finish ISR
}
Void Config_VIC ()
{
    VICIntselect = (1<<14) ; // EINT0 is FIQ
    VICIntEnable = (1<<14) ; // Enable EINT0
}
Void Config_EINT()
{
    PINSEL1 = 0x01 ; // P0.16 is EINT0
    EXTMODE = 0x01 ; // EINT0 is Edge Sensitive
    EXTPOLAR = 0x01 ; // EINT0 is rising, Edge sensitive
}
```

```
main()
{
    unsigned int cnt = 0;
    Config_EINT();
    Config_VIC();

    // default task in main()
    while(1)
    {
        cnt++;
    }
}
```