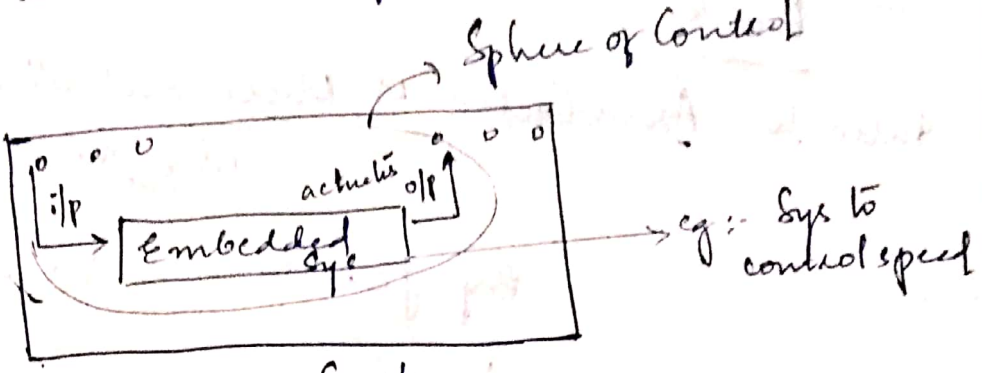
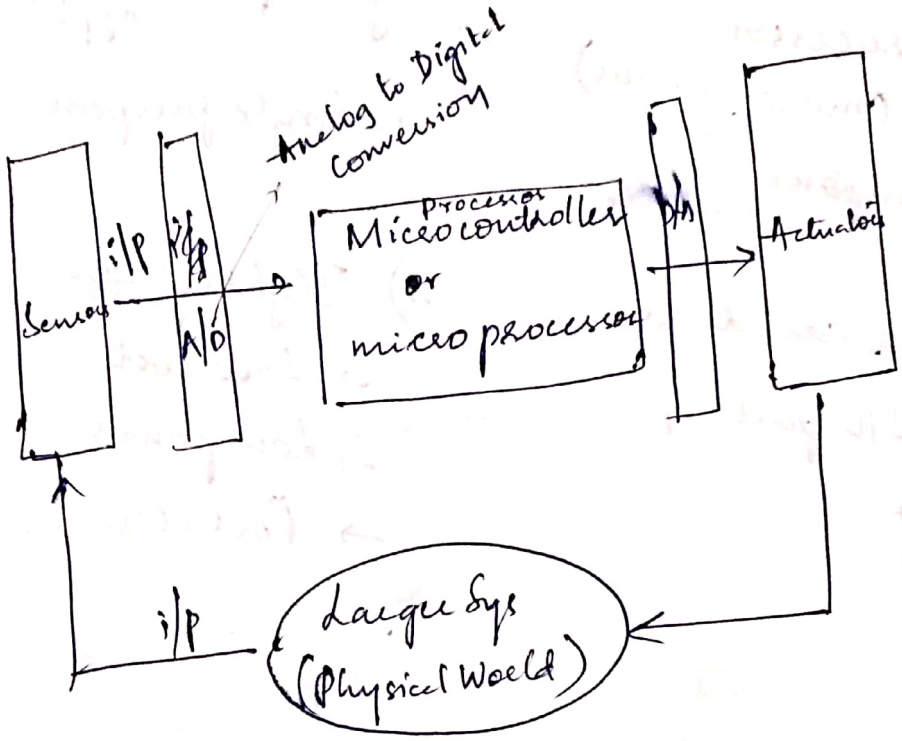
2/01/2019

Embedded Systems





Embedded System Varities. — called as Cyber physical System

↓

IOT (Embedded Sys with communication abilities) (Internet is the medium)

Book

Intro to Arm Cortex-M Microcontrollers

3rd Edn.

By Jonathan W. Valvano

Vol-1

General purpose System

Eg:- Intel processor (microprocessor)

1) Multipurpose

a) Designer can decide no of I/O port & memory

- Costlier
- Heavy
- Power inefficient

Vs

Embedded Sys

Eg:- Specific to some appln

1) Single purpose

2) Tightly Constrain
→ Low cost
→ low power
→ Portable

Real time Sys

→ Hard RTS (Stringent time constraint)

→ Soft RTS (Stringent time constraint is not required)

Vs

Embedded Sys

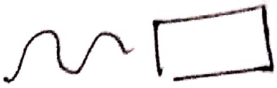
→ RTS (Stringent time constraint)

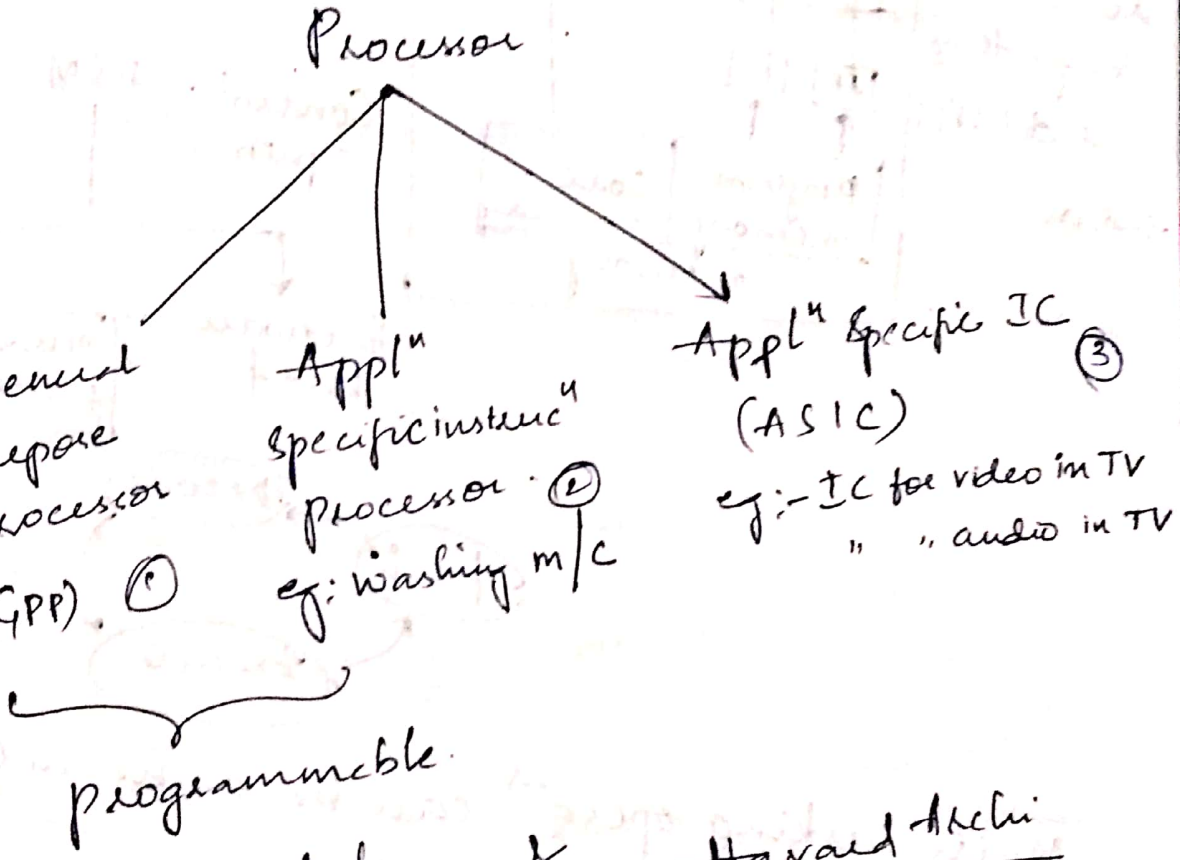
→ Not RTS (Stringent time constraint is not required)

RISC ... it is ...

Reactive Sys. Vs Transportation Sys

→ React to event → Image Processing

→ 



→ Von-Neuman Arch

Pipelining mech
↓
not possible

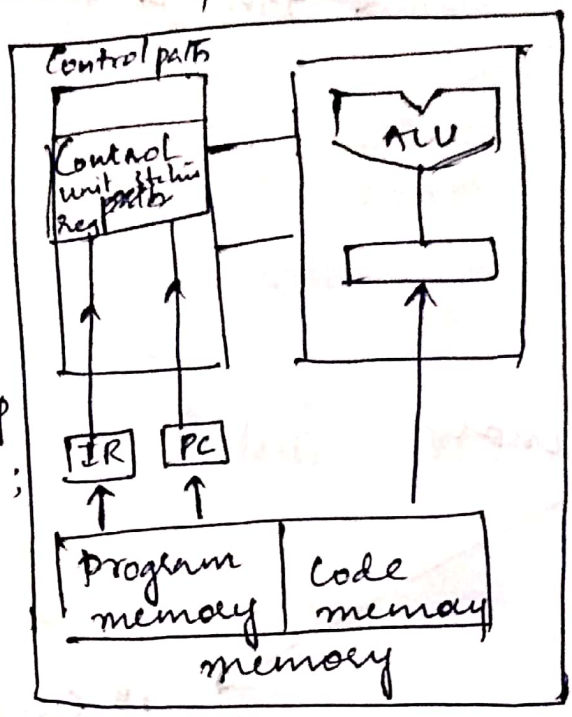
X

Harvard Archi

Pipelining mech

4/01/2019

GPP

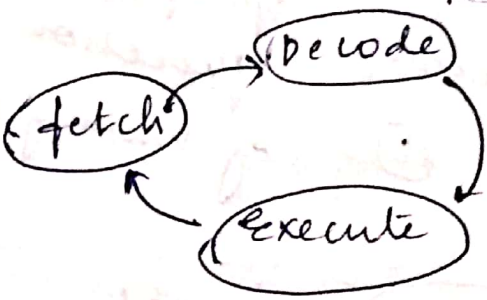



```
temp = 0 ;
for i = 1 : k
do
  temp = temp
    + data[k];
end do.
```

Data path

- Control = FSM path

Hardware based | microprog program



Multi tasking operations can be done in GPP.

GPP ①

ALU - Complex

Appln specific instruction processor ②

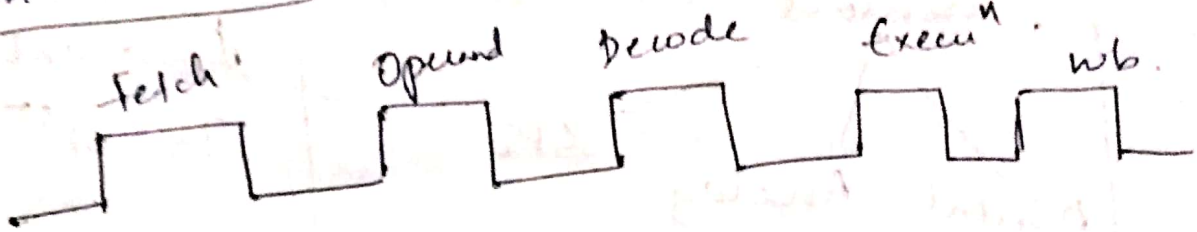
ALU - Simple

① & ② have program mem.

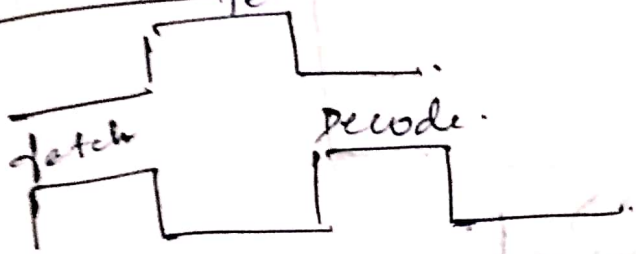
③ Doesn't have program memory

RISC → II

Von Neumann



Harvard Archi



Pipelining

ISA

CISC

RISC

Oscillator – generates sequential clock.

→ DSP → Used in Radar Communications

(Digital Signal processing)

Make a Documentation

Sensors

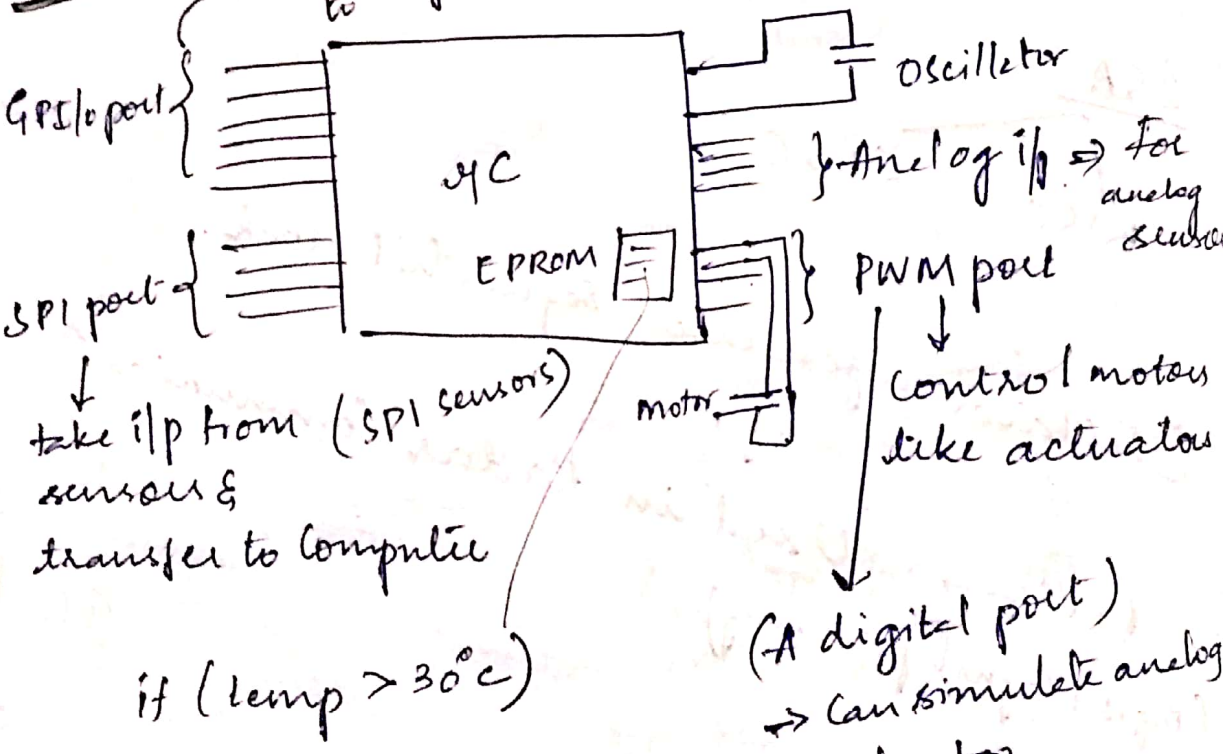
Digital Analog SPI

Last date Jan 31st

Also learn about diff types of motors

8/01/2019

(digital sensors) to digital ports



take i/p from (SPI sensors) sensors & transfer to compute

if (temp > 30°C)

control motors like actuators

(A digital port) → can simulate analog o/p too

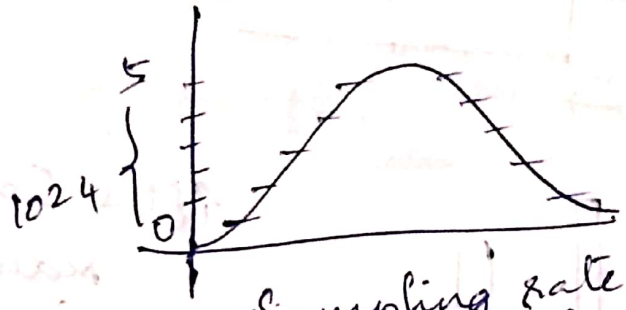
→ In a microcontroller, there is A/D converter but not D/A "

→ In DSP microprocessor, there is A/D as well as D/A converter

While using tiva c micro controller board, do these following:

1) Activate clock gating resistors
2) Set up direction (I/p or O/p)
3) Initialize port memory.

Analog Sensor



→ Info is lost in conversion

→ Sampling rate defines accuracy of conversion

Sampling rate = 2 max frequency

Resolution: Tells how you define 0 to 5V at what resolution.

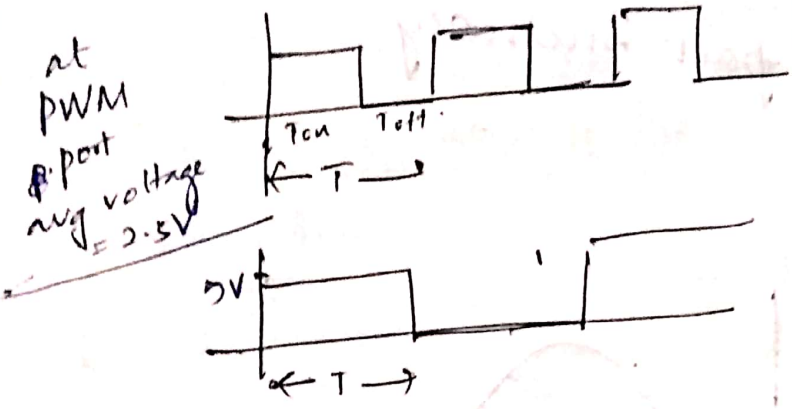
Resolution = 2^{10} = 1024

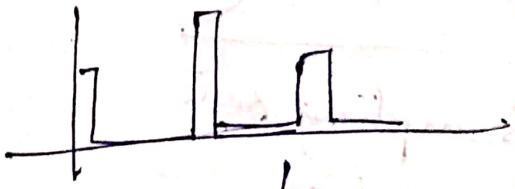
Smallest measurable voltage = $\frac{5}{1024}$ = 4.8mV

→ Sampling rate depends on ADC bit.

Pulse width Modulation (PWM)

Digital representation of Signals





On further reducing Ton time.

↓

With this we can connect LED directly without a resistor in series.

Output voltage = Pulse on time / total period.

(on + off time) * 5V

$= \frac{75}{100} \times 5V = 3.75\ V$ at PWM pins

$= \frac{25}{100} \times 5V = 1.5\ V$ "

→ With the help of this process, we can control the speed of the motor, brightness of LED's etc...

→ µC communicates with SPI port using master slave opern

SCLK = Serial clock

MOSI = Master o/p slave i/p

MISO = Master i/p slave o/p

CS — SS = Slave select

chip select

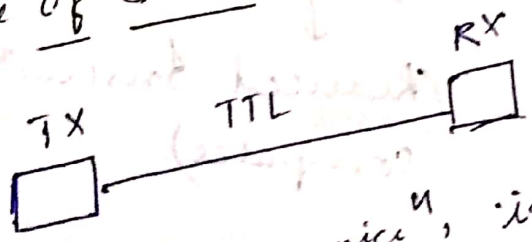
when SS line is low, then only communication takes place.

Examples of SPI sensors :-

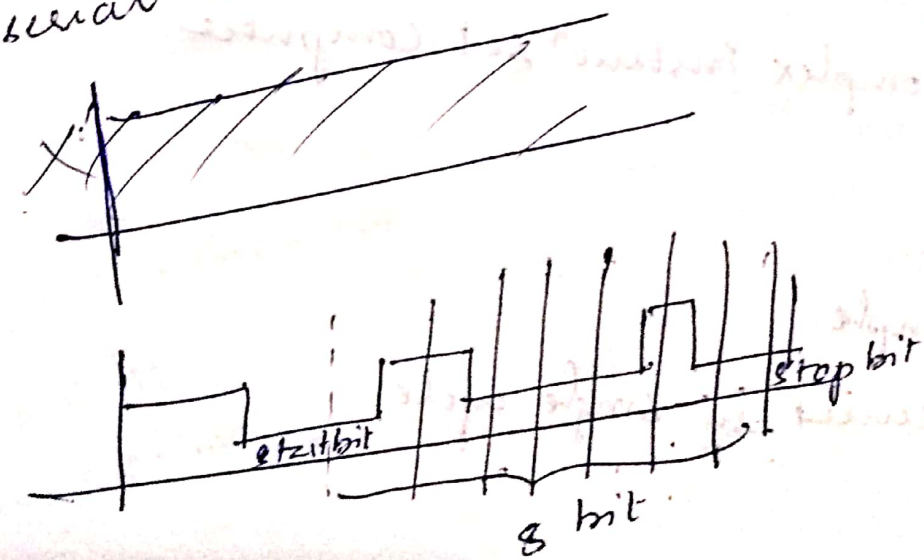
Wifi module, Zigbee module, bluetooth module.

Purpose of Serial Communication :-

TTL – Transistor transistor logic.



In serial communicn, initially TTL is high



At RX side, serial o/p is converted to parallel o/p by logic.

9/01/2019

Embedded System Design using ARM

32 bit & 64 bit
- Arm Cortex A (Appln specific)
- Arm Cortex R (Real time appln)
- Arm Cortex M (General purpose micro controller)

right now, we are using this. This is embedded in tiva C board

→ Inside Microcontroller, RISC architecture is used.

RISC
↓
(Reduced Instruction set computer)

CISC
↓
Complex Instruction set computer

RISC

→ Simple

→ Executed in single cycle.

RISC is it is

RISC

Compiler
↓ Greater complexity
Processor

CISC

Compiler
Simple
↓ Greater complexity
Processor

2 types of instructions in ARM.

① Thumb instruction 16 bit
② ARM " 32 bit

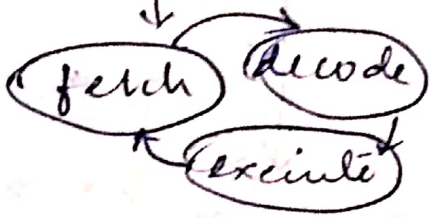
ARM Microcontroller
→ low cost & high form factor (Die area)
→ Nanowatt micro controller.
→ On board debugger is present in it. (JTAG)
Allows developer to see what's happening inside µC

→ It is Adv RISC M/C.

ARM
Adv RISC m/c

Extra-features

- Instruc^n ← Small no. of instruc^n classes; Instruc^{ns} are very simple & exec in single cycle
- Pipeline → can achieve 1 i/cm
- Register → Has many general purpose regis
- Load store archi
  - load from mem.
  - perform opns.
  - store the result.



ARM is not a pure ~~RR~~ RISC archi

⇒ variable cycle execution for certain instruc^n

- load / store multiple

→ Inline barrel shifter

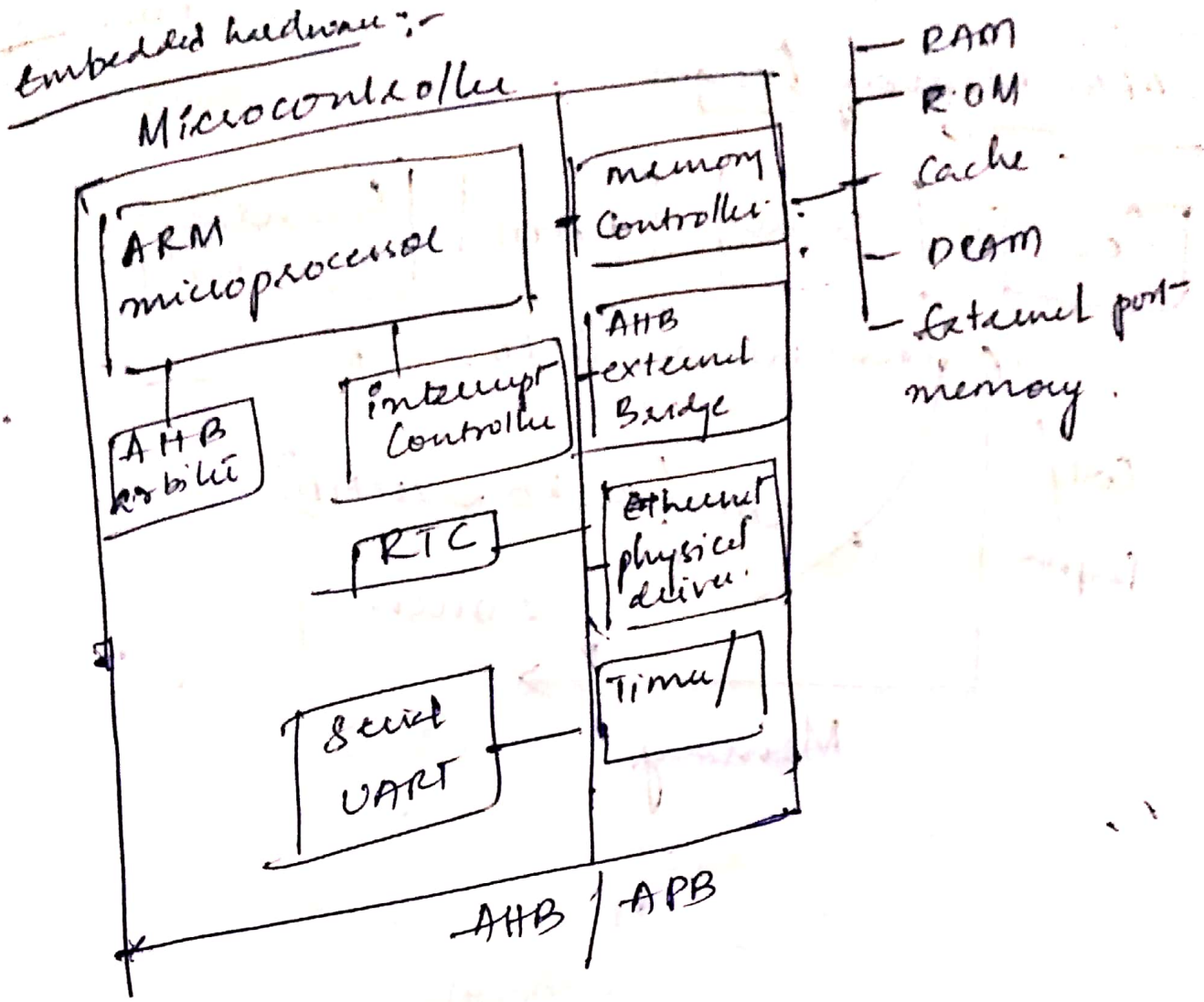
→ Thumb instruc^n – 16 bit (High code density)

→ Conditional Execution

- Add if carry / overflow / zero

→ Enhanced instruc^{ns}. (faster executions)

Embedded hardware :-



→ AMBA Architecture
— On chip bus archi
↳ Advanced Microcontroller Bus architecture

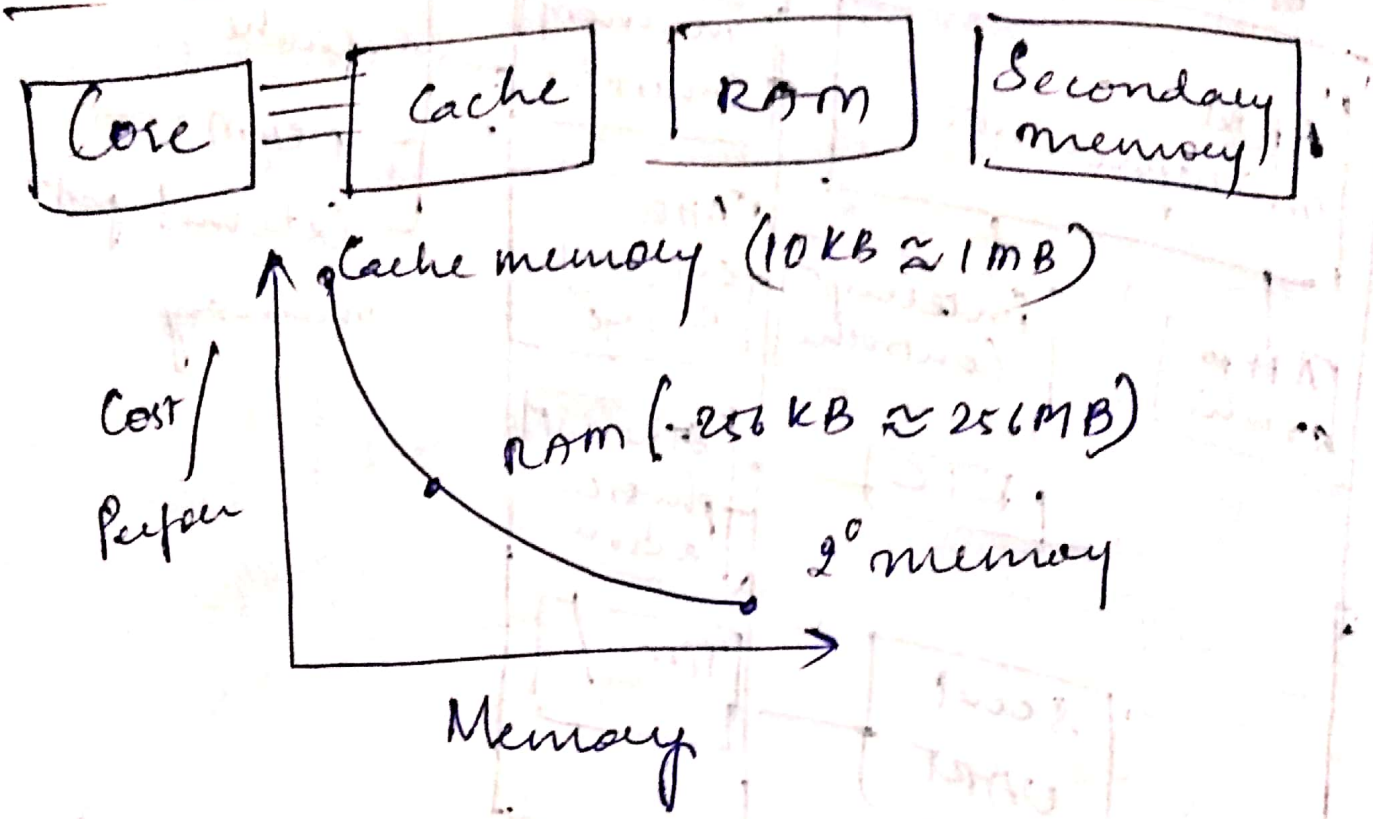
1996

APB — (Advanced peripheral bus)

AHB (Advance high performance bus)

PCI
External bus archi

ARM Memory Design

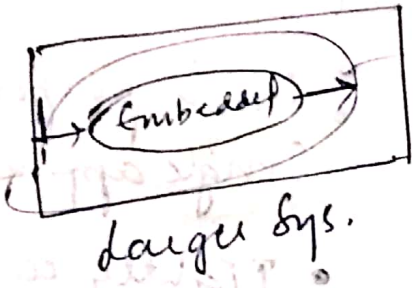


NPTEL 39:14

ES

Intro to ES design

ES :- An ES is a sys where a µc based programmable sys is embedded in a larger sys

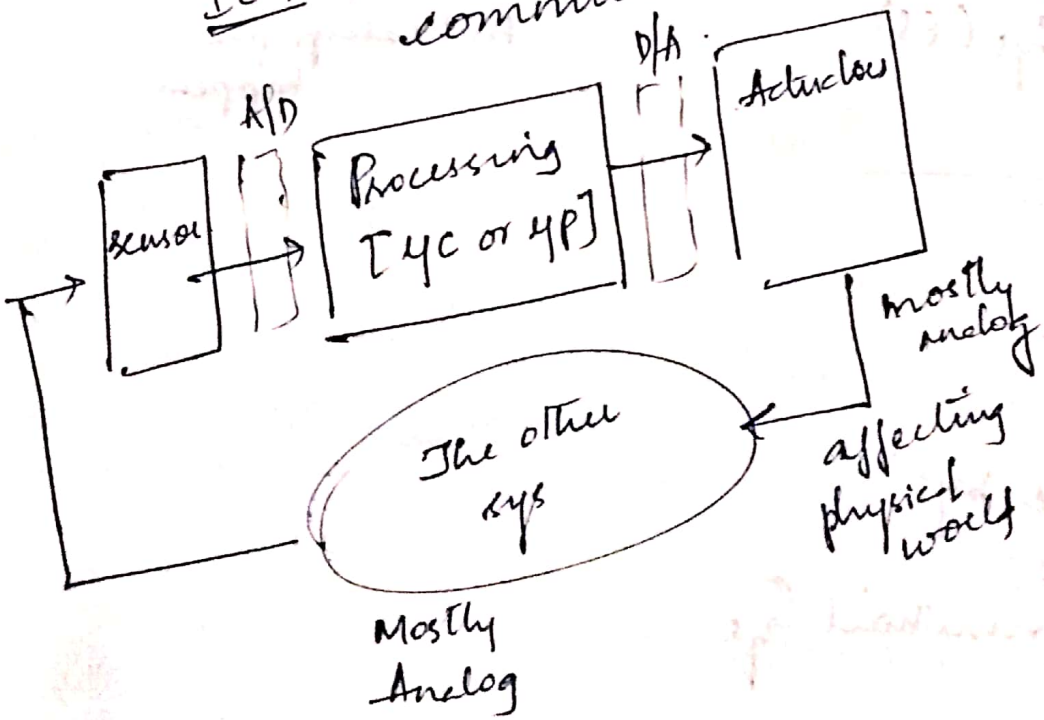


→ Automatic cars, aeroplanes, washing m/c. all have embedded controllers.

Cyber-Physical Sys :-

Computer — Can be used

IOT : set of sys spread around & communicate over internet.



Sensor can directly sense data in digital form otherwise, A-D converter converts it.

General purpose sys

Vs

Embedded Sys

- Single purpose Single application
- Tightly constrained
  - Low cost
  - Low power
  - Portable
  - Sometimes real time

predefined appln

→ More powerful

→ Consumes more power

→ ~~predefined appln~~

→ Real time Sys (RTS) → Hard RTS & soft RTS → stringent time constraint → Quality of service But catastrophe happens

→ Embedded Sys (ES)

RTS or may not

→ Reactive Sys

Vs

Transformational Sys

Reactive

React to an event

Example

→ Img processing sys
  • Segment
  :

Challenges (optimizations)

→ Cost
  ↔ NRE cost
  Non recurring Engg cost
→ Size
→ Power
→ Performance — Latency, throughput
  ↓
  0.25 sec.
  Camera A = 4 pic/sec } throughput
  Camera B = 8 pic/sec. }
→ Flexibility
  ph: 4G+5G ✓
  ph: Only 5G

Lecture 2

Intro to Processors

Other Constraints

→ Time to prototype:- Time taken to build
Need to have a quick prototype

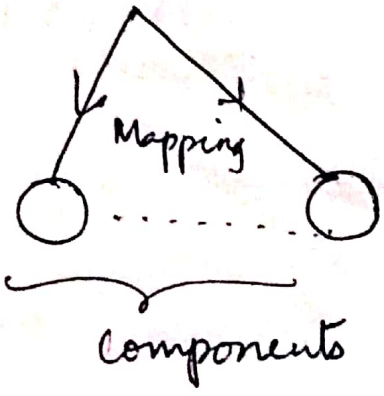
→ Time to Market:-

→ Maintainability:- How easy to maintain
How quickly can I recover from a problem.

→ * Correctness; Safety :-
Correct functionally
Correct with timing

Design Flows

~~Hardw~~ Requirements & Specifications } can make entirely software or entirely hardware



Lec 3

① ASIC — Appl^n specific IC → cannot change functionality

② Single purpose processor.

↳ Programmability

ISA : It is what differentiates b/a architectures of processors.

→ ISA is large for single PP.

ISA is small for ASIC

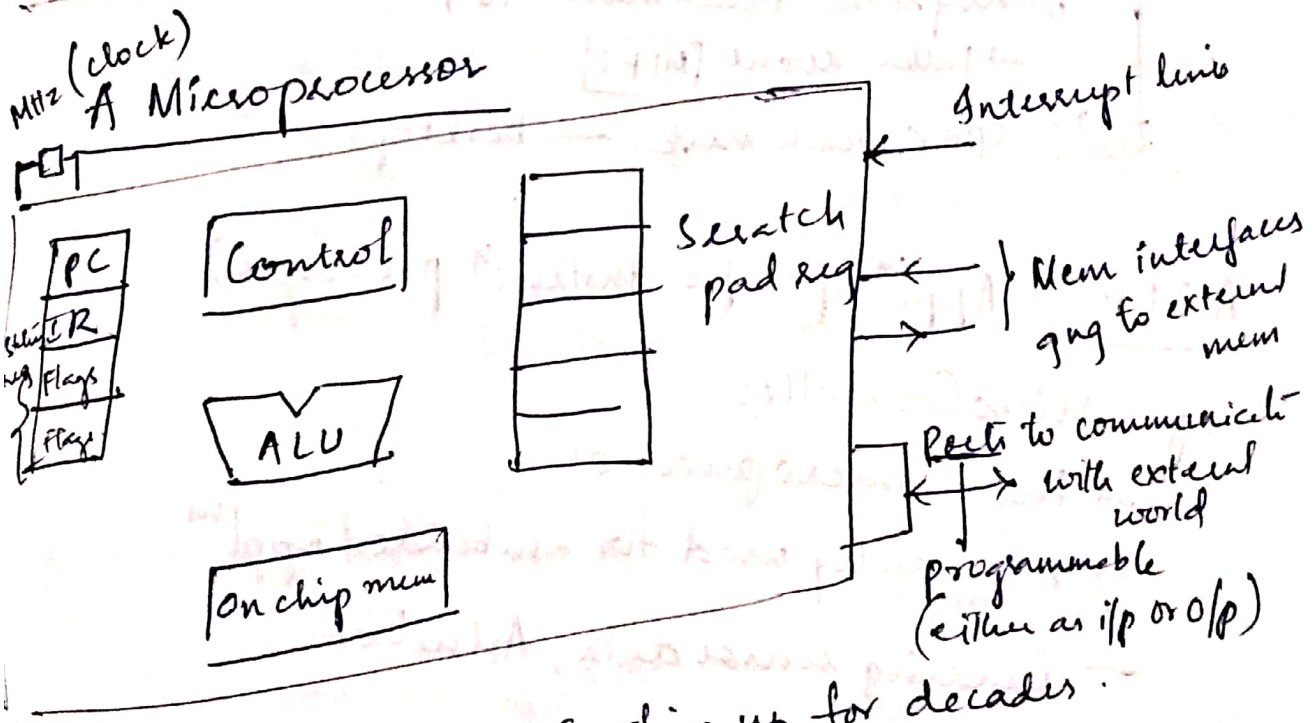
GPP

eg: Spark m/c, Pentium

Embedded Sys

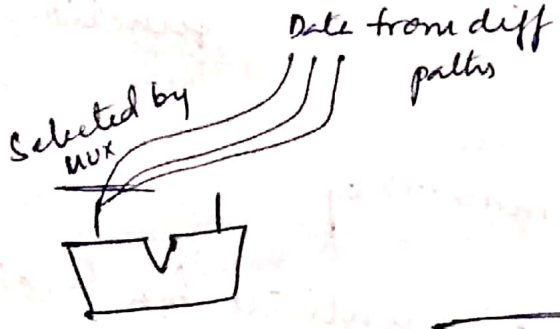
GPP is represented by micro processor

also GPP



→ Clk speed has been shooting up for decades.

→ Controller activates instrucns.



* → How do you select processor for an ES?

1. Speed
2. Power
3. Size
4. Cost

} Technical specs.

5. Familiarity (with ISA)

→ clk speed (faster the clk, faster instrucns are executed)

→ Instrucn's e.g. per sec

→ 1. Dhrystone benchmark 1984
→ Talks about MIPS

→ 2. SPEC benchmark — Desktops

ASIP (Applⁿ specific Instrucⁿ processor)

eg:- MicroController
→ Has a microprocessor.
→ Essentially used for embedded applns.
→ Reading sensor data, Actuation
→ Deals with events
→ Used in disk drives, digital Camera, Hardware ckts....

ISA is simple for Microcontroller

→ Has • timers, A/D conversion, Communic^n ports,

• On chip memory — Program Memory
Data memory

• Allows you to have directly access

• Specialized instruc^ns for bit manipul^n

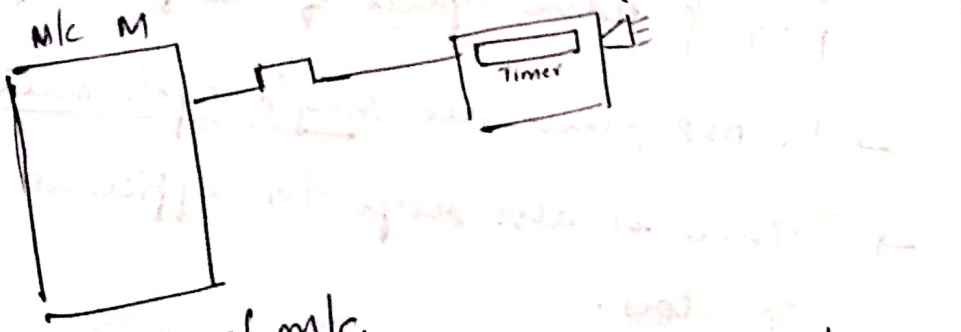
& other low level compute^ns

not there in µprocessor But present in µcontroller

Timer — Down Counter (when it reaches '0' then gives a pulse)

→ Very Interesting Embedded Appl^n requires "Watch dog Timer".

→ A timer watches out for an event to occur.



Aim:-

To watch health of m/c

Every 15 sec, µ must get a pulse.

The timer should have 15 sec

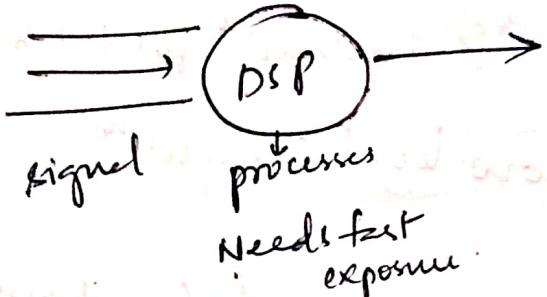
when it reaches 0 & pulse doesn't arrive, then it raises alarm.

→ If the m/c is healthy, by the time it reaches 0, pulse arrives & timer again sets back to 15 sec

⇒ Expecting some event to occur.

Digital Signal Processor :— (DSP):

→ Used for signal processing.



→ DSP has multiple execution units.

→ Multiply - Accumulate instruction.

Common Operation in DSP :—

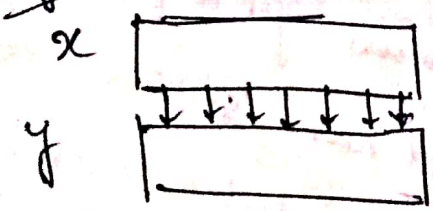
$$y = \sum_{i=1}^{N-1} x_i a_i$$

DFT, FFT } have operations of the form $y = \sum_{i=1}^{N-1} x_i * a_i$

→ In DSP, there are single cycle multipliers.

→ There is also scope for efficient vector operations.

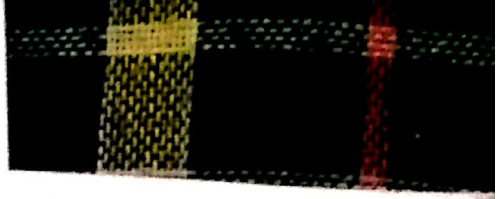
eg:-



$z = x[i] + y[j]$

// add

RISC as it is simpler.



ARM < Thumb nail mode (work in 32 bit)
Normal Mode

→ For DSP's, study TMS.

→ Earlier, we had μprocessors as chips.

→ Now, we are buying intellectual properties (IP's)

↓

Predesigned cores (code)

eg: Synthesizable VHDL (say)

↓

put it in your design tool

(customize data path)

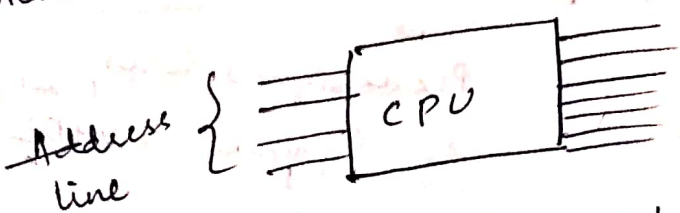
add couple of hardware or instructions.

14/01/2019

Data width

Arm – 32 bit

Mem size – 16 bit (2 Byte)

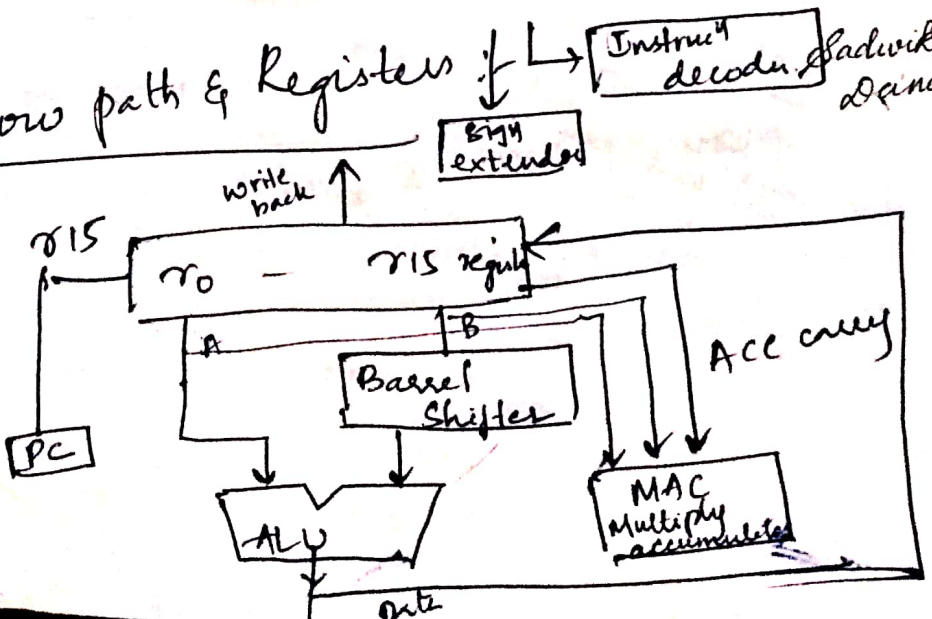


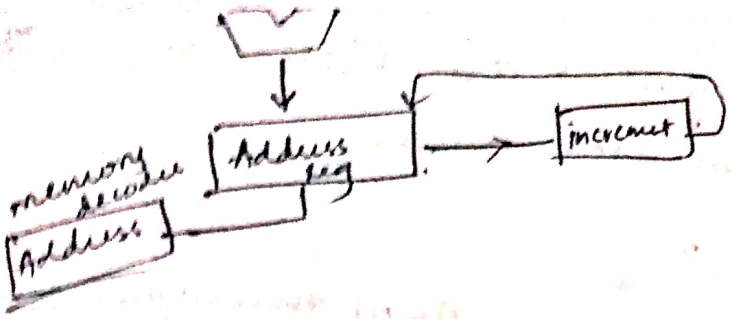
Instruction Size — 8 bit mem 16 bit mem 32 bit mem

Arm 32 bit — 4 cycles is necessary to fetch ~~32 bit~~ 8 bit.

	8 bit	16 bit	32 bit # mem
Instru" size:			
Arm 32 bit	4 cycles for 32 bit (to fetch)	2 cycles for 32 bit	1 cycle for 32 bit
Thumb 16 bit	2 cycles.	1 cycle	1 cycle.

Data flow path & Registers





* → Sign extendor converts 8 or 16 bit to 32 bit

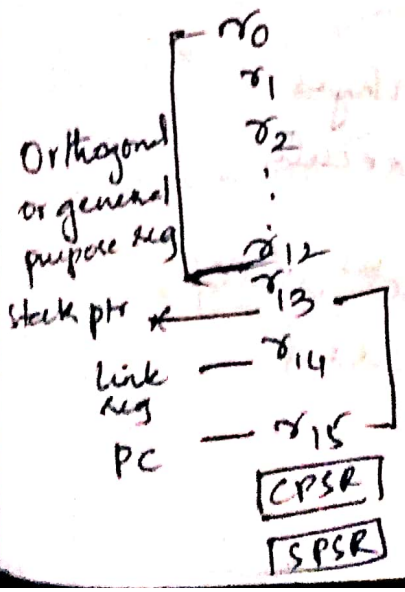
* **Barrel Shifter** :-
Does logical operations.
eg. logical AND, OR, right shift, left shift

* **Program Counter** :- Stores address of next instruction to be executed.

* **Instruction Decoder**:-
ADD, or MULTIPLY... is decoded by this instruction

* **Different Set of register** :-
16 registers inside the Arm microprocessor that are visible to a programmer.

CPSR.
SPSR



Special fn registers.

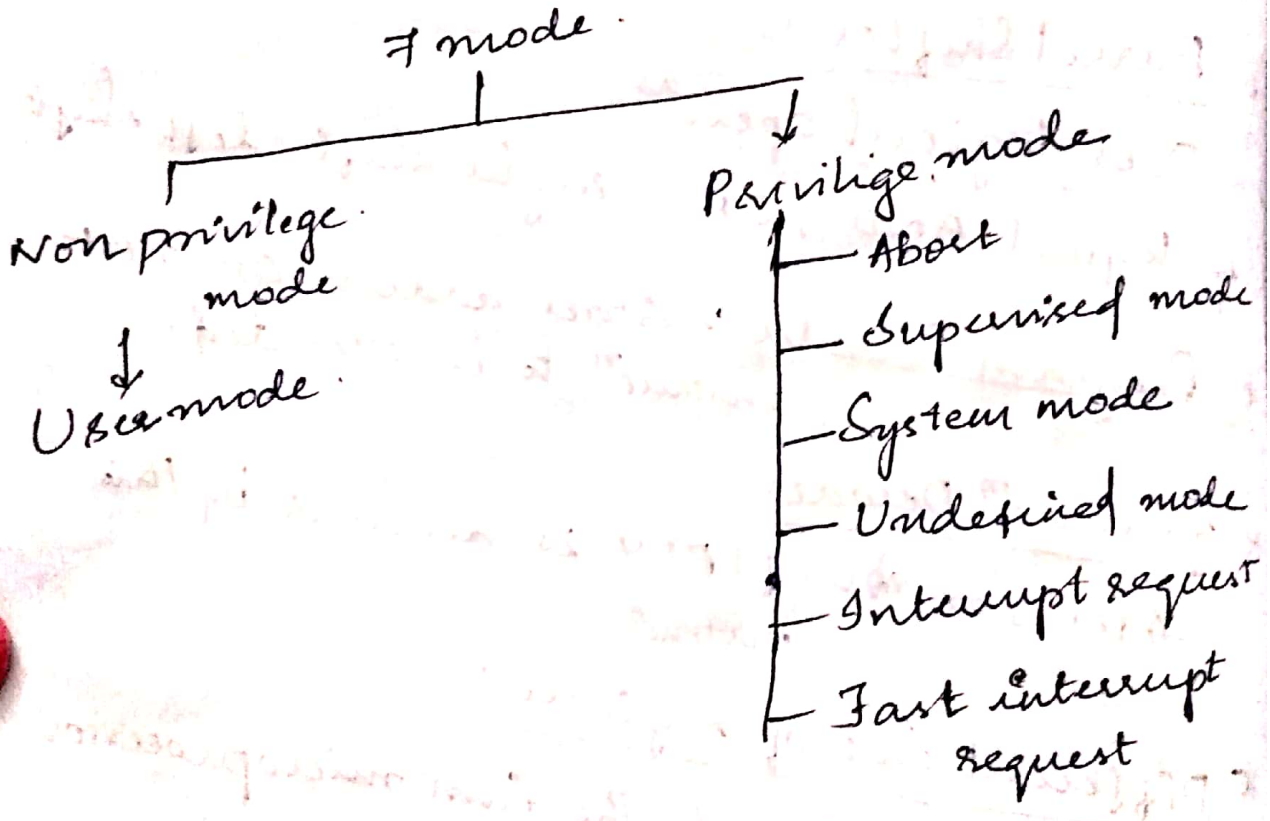
Current program status reg
Saved program status reg

→ All these registers are visible to developer in user mode.

What is mode?

There are 7 modes in ARM microprocessor

~~Each mode has 18 registers (set)~~



Abort – Invalid memory access.

Supervised – When you reset pc, it goes into this mode.

System – Have all the priviliges Read / write / execute

Undefined – Invalid instructions which cannot be decoded

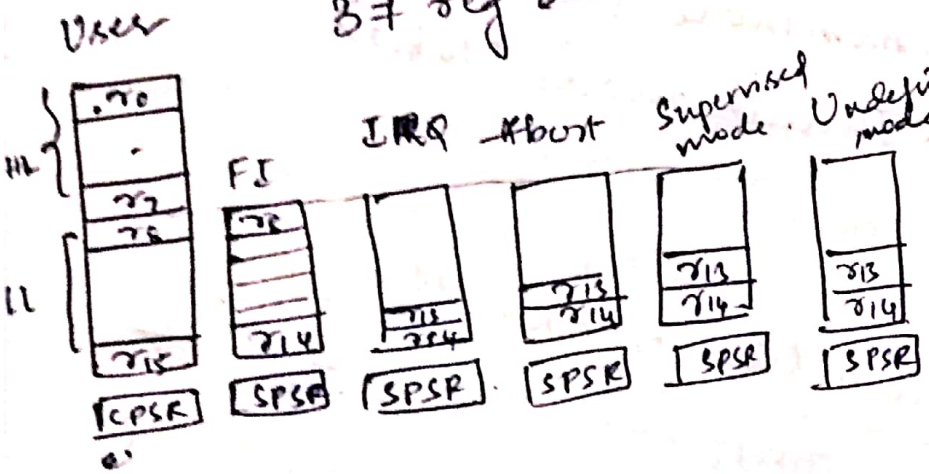
Fast Interrupt :- Interrupt request
- Interrupt request

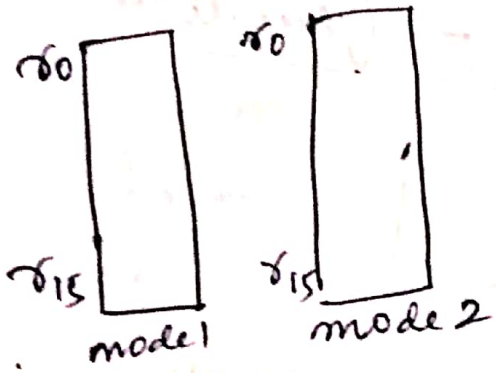
User — $r_0 - r_{15}$ & CPSR

$r_0 - r_7$ — high level reg

$r_8 - r_{15}$ — low level reg

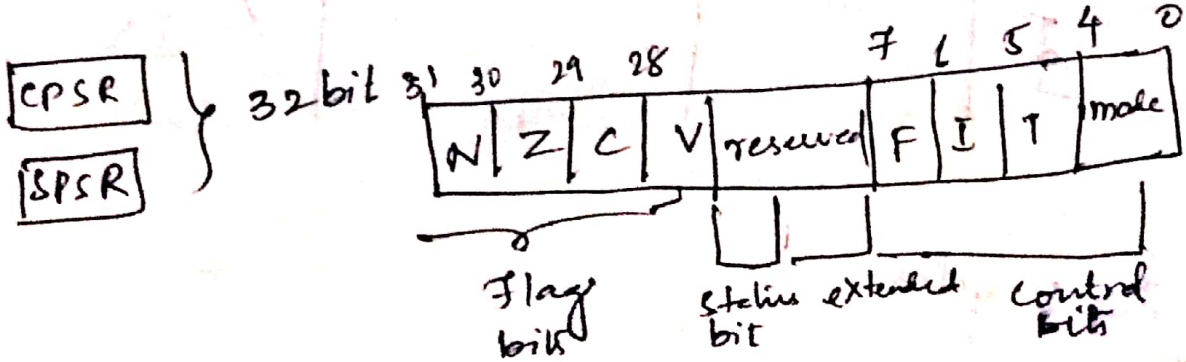
37 reg set comes from all 7 modes





SP → save the info. into reg $r_0 - r_{12}$

SPSR → Processor status



N - Negative flag

Z - when ALU generates 0 result

C - when ALU generates carry.

V - Overflow

F - fast interrupt req

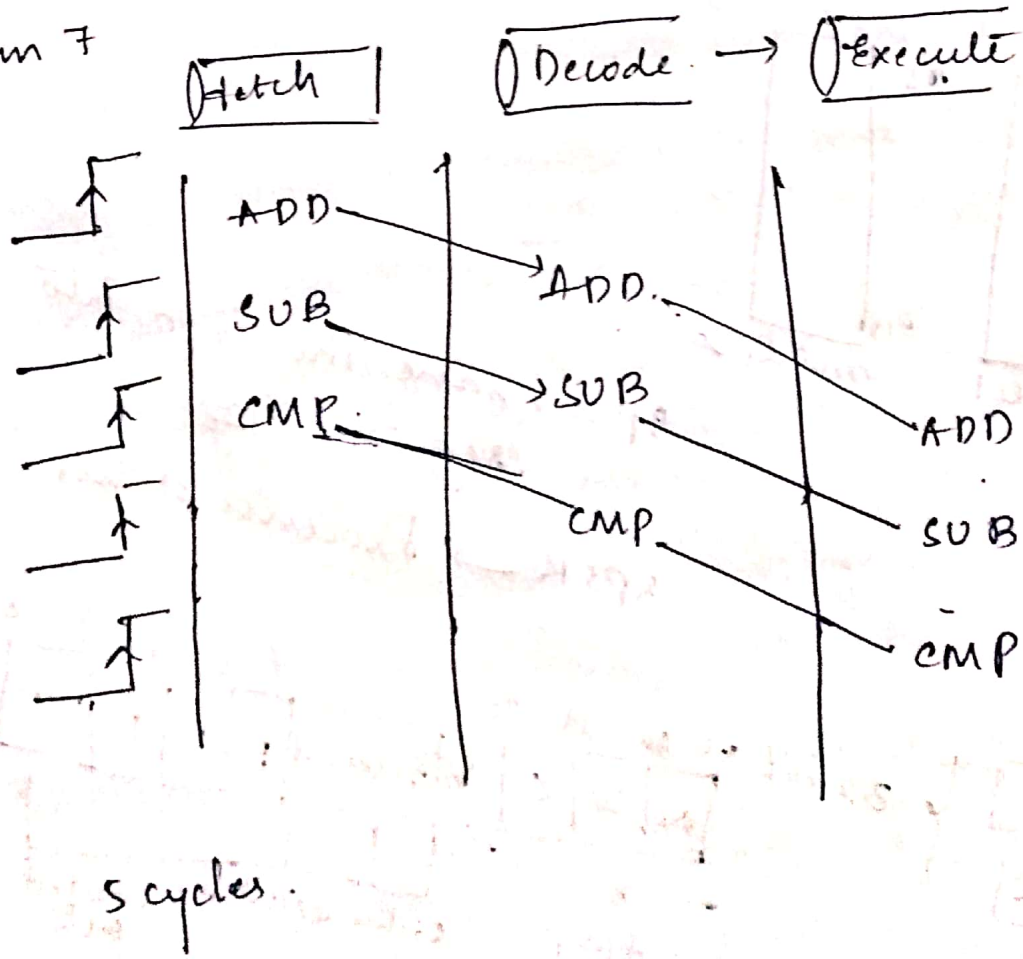
I - Interrupt req

T - < Arm mode (32 bit)
    < Thumb mode (16 bit)

25/01/2019

Pipelining Architecture of ARM Microcontroller family

Arm 7



5 cycles.

ARM 9 → Has 5 stage pipelining architecture

Fetch → Decode → Execute → Memory → Write

ARM 10 → Has 6 stage pipelining

Fetch | Issue | Decode | Execute | Memory | Write

← Latency ↑ →

Instruction set of ARM Microcontroller :-

- Data processing instruction
- load/store instruction
- Program status register instruction
- Software interrupt instruction
- loading constant instruction
- Branch instruction
- Conditional execution instruction

1. Dataprocessing instruction :-

mov {s}.

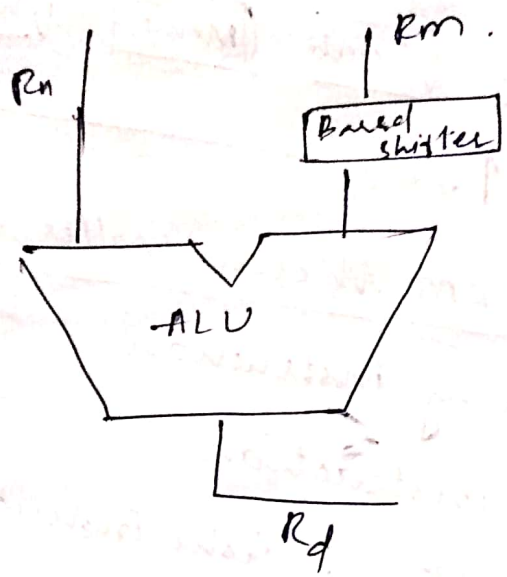
{0-31} 4at cont.

Syntax

<instruction> <condition> <s> <Rd> <N>

Memonics — optional. — destination reg.

eg:- r0 = 5;
r1 = 7;
mov r0, r1; r0 = 7
r1 = 7



Barrel shifter preprocessing commands

- LSL (logical shift left)
- LSR (logical shift right)
- ASR (Arithmetic shift right)
- ROR (Rotate right)
- RRX (Rotate right extended)

movs r5 r7 LSL #1;

0000 0111
0000 1110

r5 = 14 r7 = 7

CPSR : N Z C V I F T

N=1 (Negative result) Z=1 (Zero)

eg:

r_7 = 0000 0000

r_5 = 1000 0100

movs r_7 r_5 LSL #1

1000 0100

0000 1000

→ C = 1 — carry occured

→ NZCV... are used for conditional executions

eg:- Add if carry occurs.
Sub if carry occurs.
Add if zeroflag is high.

Conditional Instruction :-

↓

2 letter neumonics

AL = Always
EQ = Equal
GT → Greater than
LT → less than.
.
.
.
BAL = Branch always.
BEQ = Branch if equal
BNE = Branch not equal
B = Branch
.
.

if (a == b)
go to test;

r0 = a
r1 = b
CMP r0, r1
BEQ test → label

28/01/2019

Dataprocessing & conditional instruction can only update the flag of program status register, only if 's' suffix is with instruction.

ARM Instruc" < Thumb Instruc" (16 bit)
ARM " . (32 bit)

Thumb Instruc" — Cannot execute conditional support execution

ARM " — Can support " .

Thumb :- Syntax :-
<Instruction> <'s'> <Rd> <N>
destine" reg

ARM :- <instruction> <Cond"> <s> <Rd> <N>
↓ :.
ADDEQ

ARM

→ Takes more memory

⇒ Dense

Thumb

→ Takes less mem.

⇒ Less dense

GCD

```
while (a != b)
{ if (a > b)
    a = a - b ;
  else
    b = b - a ;
}.
```

Code conversion into Thumb & ARM Instruc^n.

Thumb (16 bit)

7×2 = 14 bytes

```
gcd
    CMP r0, r1 ;
    BEQ complete ;
    BLT lesthan ;
    SUB r1, r1, r0 ;
    B gcd ;
lesthan
    SUB r0, r0, r1 ;
    B gcd
complete
```

ARM (32 bit)

4×4 = 16 bytes

```
gcd
    CMP r0, r1 ;
    SUBGT r0, r0, r1 ;
    SUBLT r1, r1, r0 ;
    BNE gcd ;
```

Arithmetic Instruction :-

Syntax :-

<Instruction> <condn> <S> <Rd> <Rn> <Rm>

Rd → destination reg
Rn, Rm → Source reg

SUB Rd, Rn, N ; ⇒ Rd = Rn − N

ADC (Add with carry)

ADC Rd, Rn, N ; ⇒ Rd = Rn + N + carry

ADD Rd, Rn, N ; ⇒ Rd = Rn + N

SUB Rd, Rn, N ; ⇒ Rd = Rn − N

RSB (Reverse subtract)

RSB Rd, Rn, N ⇒ Rd = N − Rn ;

RSC (Reverse subtract with carry)

RSC Rd, Rn, N ⇒ Rd = N − Rn − ! (carry flag)

r0 = 0x0000 0001 ;

SUB r0, r0, #1 ; (r0 = r0 − 1)

```
  0x0000 0001
     1111 1110
              |
r0 = 1111 1111   2's complement
  (1)1111 xxxx
  0x0000 0001
+ 0x1111 1111
```

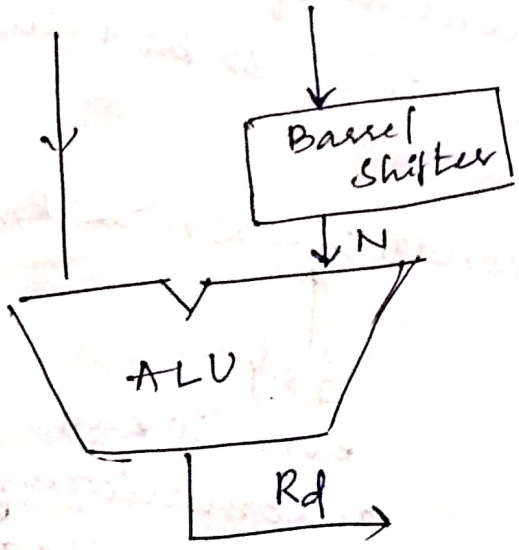
Carry = 1
Result = 0
⇒ Z & C bits are set. if you use SUBS as instruc^{n}.

Logical Instruc^{ns} :-

We update SPSR registers while using logical instruc^{ns}.

Logical Instruc^{n}s :-

~~ADD~~ AND ⇒ Rd, Rn, N ; ⇒ Rd = Rn & N
ORR ⇒ Rd, Rn, N ; ⇒ Rd = Rn | N
EOR ⇒ Rd, Rn, N ; ⇒ Rd = Rn ^ N
BIC ⇒ Bitclear
↳ Rd, Rn, N ; ⇒ Rd = Rn & (~N)



r0 = 0x02040608 ;
r1 = 0x10305070.
ORR r0, r0, r1 ⇒ r0 = r0 | r1
r0 = 12345678

r0 = 0x1111
r1 = 0x0101

BIC r0, r0, r1 ; r0 = r0 & (~r1)

Sensors
1) SPI
2) Simple digital
3) Analog
4) PWM

29/01/2019

Comparison Instruc^{n} :-

CMP ⟶ comparison
CMN ⟶ Comparison negation
TEQ ⟶ Test the content of two 32 bit reg value
TST ⟶ Test the bits of 32 bit value.

Syntax :-
<instruction> <condn> <Rn> <Rm>

{S}
even it can automatically update the flag of program status reg

CMP r0, r1 ;⟹ r0 − r1
↓
update flag

CMN r0, r1 ; ⇒ (r0 + r1)

↓ Arithmetic Instruction.

TEQ r0, r1 ; r0 ^ r1 update the flag reg

TST r0, r1 ; r0 & r1 update the flag reg

Multiply Instruction in ARM Microcontroller

Syntax:-

MLA {condn} {S} Rn, Rm, Rs, N

Used in "DSP"

⇒ Rn = (Rm * Rs) + N

MUL {condn} {S} Rd, Rn, Rm ;

Rd = Rn * Rm

Signed Multiply Instruction:-

SMLAL → (Signed multiply Accumulate long)

SMULL

Unsigned Multiply Instruction:-

UMLAL (Unsigned multiply accumulate long)

UMULL

UMULL R_d, R_0, R_s, N ;

[R_d, R_0] = R_s * N

↓ Higher bits ↓ lower bits

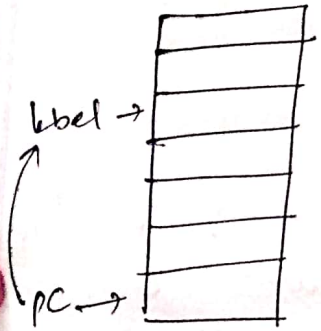
Branch Instruction in ARM Microcontroller :-

B {condn} label

BL {condn} label → Stores address of next instruction in the link reg.

BX {condn} label

BLX {condn} label Rm → Changes the mode (Thumb to ARM OR ARM to Thumb)



BX IA → LR ⇒ Address of next of current prog execution

PC = label.

eg:- B forward

ADD R0, R1, #2;

ADD R2, R3, #4;

forward

SUB R0, R1, #1;

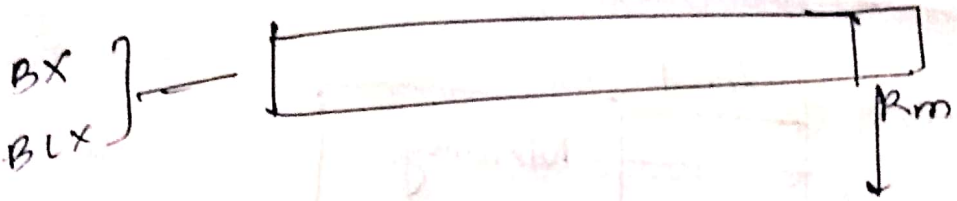
Backward

ADD R4, R5, #1;

SUB R6, R7, #4;

B Backward

ARM Instruction mode - 32 bit
Thumb Instruction mode - 16 bit



0 ⇒ ARM mode
1 ⇒ Thumb mode

Analog

Analog temp sensor

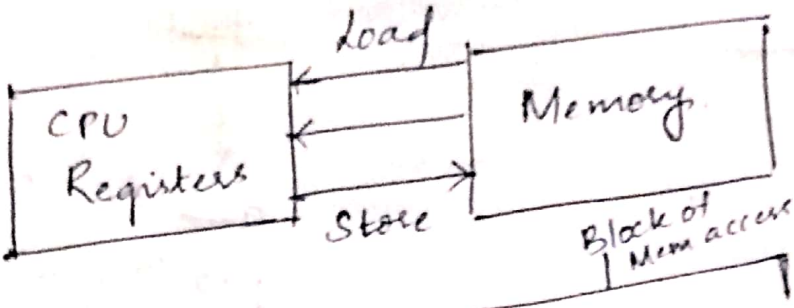
- Pressure sensor
- Sound Sensor
- Light Sensor
- Accelerometer

Digital

Digital temp sensor

1/02/2019

Load/Store Instruction in ARM Microcontroller



Syntax:

LDR {condn} SB/B/H/SH, Rn, Addressing mode

STR {condn} B/H, Rn, Addressing mode

Addressing Mode:-

1. Preindex with write back.
2. Preindex
3. Postindex

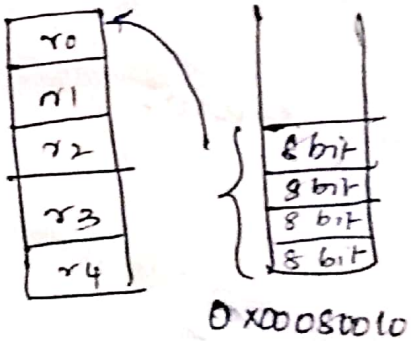
Single reg LDR/STR Instruction:-

LDR, STR, LDRB, STRB

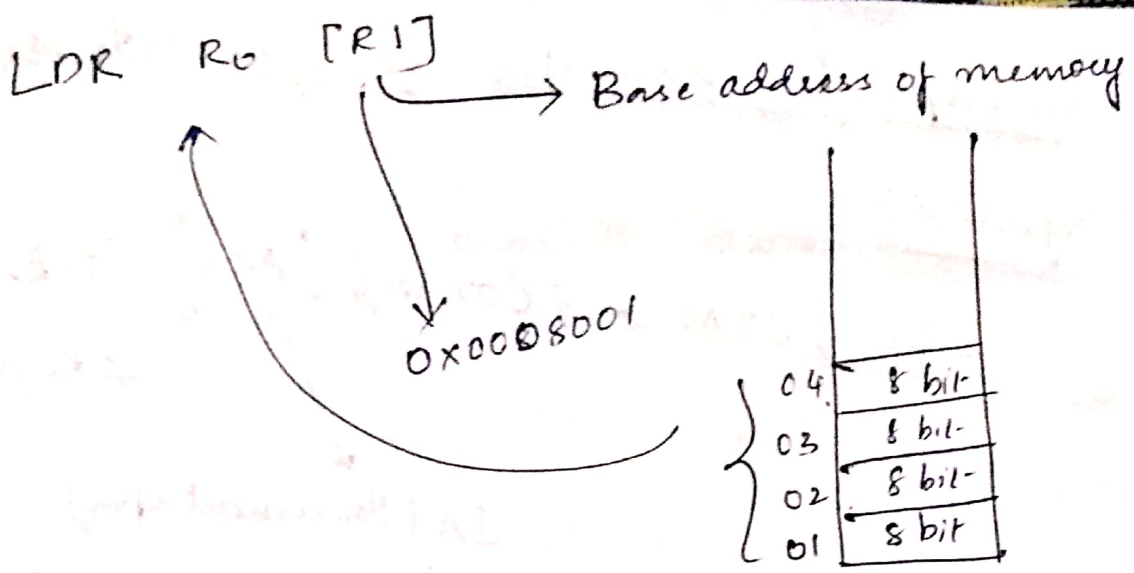
1 Byte transfer

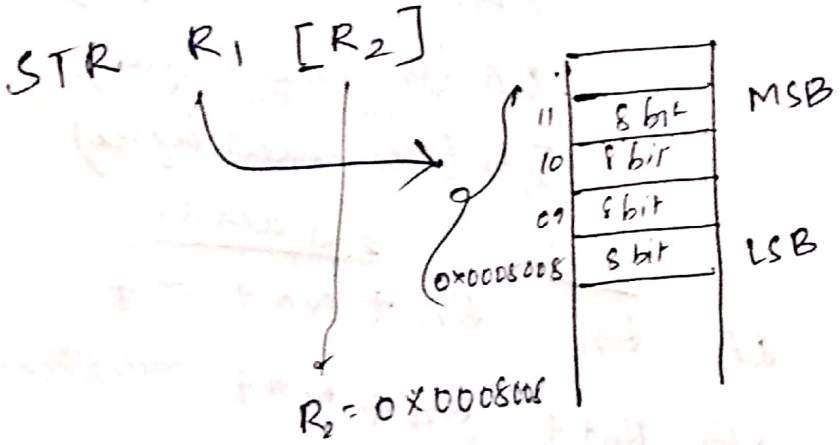
STRH, LDRH.

halfword (2 bytes) 16 bit



LDRSB → Signed 8 bit mem transfer





Little Indian
Big Indian

Example.

Indexing Mode	Data	Base Address	LDR R0 [R1, #4]!
Preindex with write back mode	{BaseAddress + Offset}	BaseAddress + Offset	LDR R0 [R1, #4]
Preindex	[BaseAdd + Offset]	No change	LDR R0 [R1], #4
Postindex	[Base]	BaseAddress + Offset	

Multiple Reg transfer load/store Instruc

Syntax:-

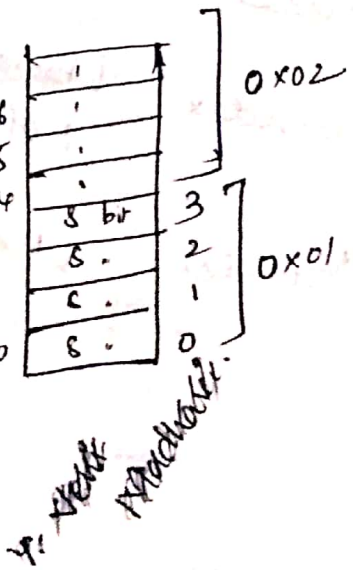
<LDM/STM> {Condn} {Addressing mode} {Rn}{!} <Registe

IA (Increment after)
IB (Increment before)
DA (Decrement after)
DB (Decrement before)

	start	end address.
IA	Rn	Rn + N*4 − 4
IB	Rn+4	Rn + N*4
DA	Rn − N*4+4	Rn
DB	Rn − N*4	Rn − 4

r0 = 0x0008 0010
r1 = 0x0000 0000
r2 = 0x0000 0000
r3 = 0x00000000

LDMIA r0{!} [r1 − r3]
r1 = r0 (0x01)
r2 = r0 + 4 (0x02)
r3 = r0 + 8 (0x03)



STMIB r_1!} $[r_3 - r_6]$

↓

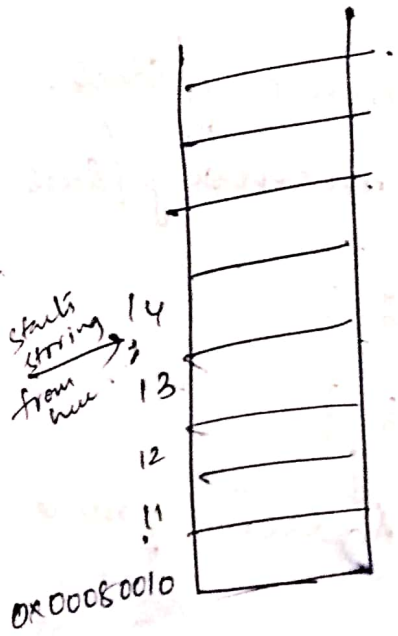
ptr to Base Address of Mem.

$r_3 = 0x00010001$

$r_4 = 0x00100100$

$r_5 = 0x01000110$

$r_6 = 0x10001000$



18

18 + 4

18 + 8

Doubt ??

4/02/2019

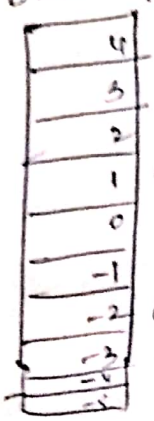
Stack Opera" in Micro Controller

Load/Store multiple register instruc" can carry out stack opera"

push → Store the data in memory/Stack
pop → get the data from memory/Stack

push → performed by STM { }
pop → performed by LDM { }
↓
Addressing mode

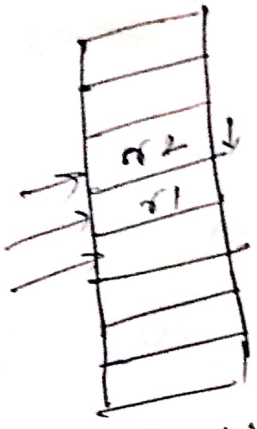
Stack Opera" Mem



STMEP sp{!} <r1, r2>
↓
Full Addressing

r1 = 0x0000 0002
r2 = 0x0000 0001
sp = 0x0008 0014

FD (Full descending) LDMED
FA (Full Ascending) LDMFA STMFA
ED (Empty descending) LDMED STME
EA (Empty descending) LDMEA STMEA
↓ ↓
pop push



STMFD SP!, {r1, r2}

r1 = 0x0000 0002

r2 = 0x0000 0001

SP = 0x0008 0014

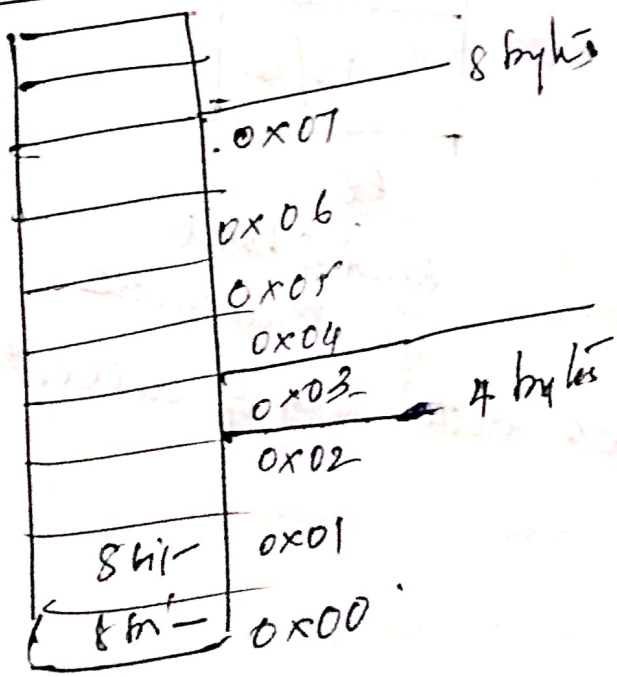
r2 — stored first
then r1

ATPCS

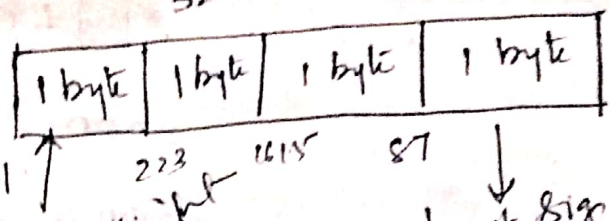
↳ ARM — Thumb procedure call standard

1) When processor is changing its state.
2) Interrupt Occurs
3) Fn call.
4) Process saves current progeam status

Big Endian & Little Endian

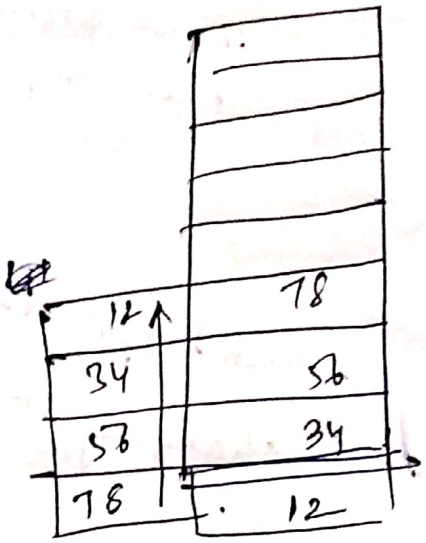


32 value



12 34 56 78

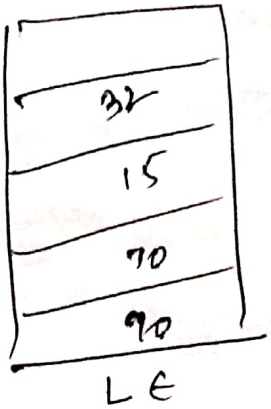
MSB LSB



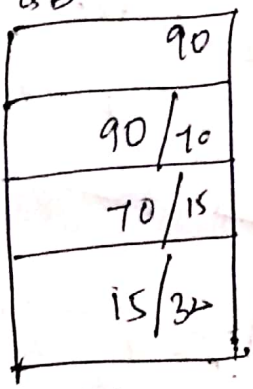
Little Endi Big endian

int a = 32 15 70 90 ;

MSB LSB



LE



BE

b = 90 70 15;

b = a

b = 32 15 70 90

MSB LSB

Read & Write operations are more

ARM µC follows both big & little Endian formats.

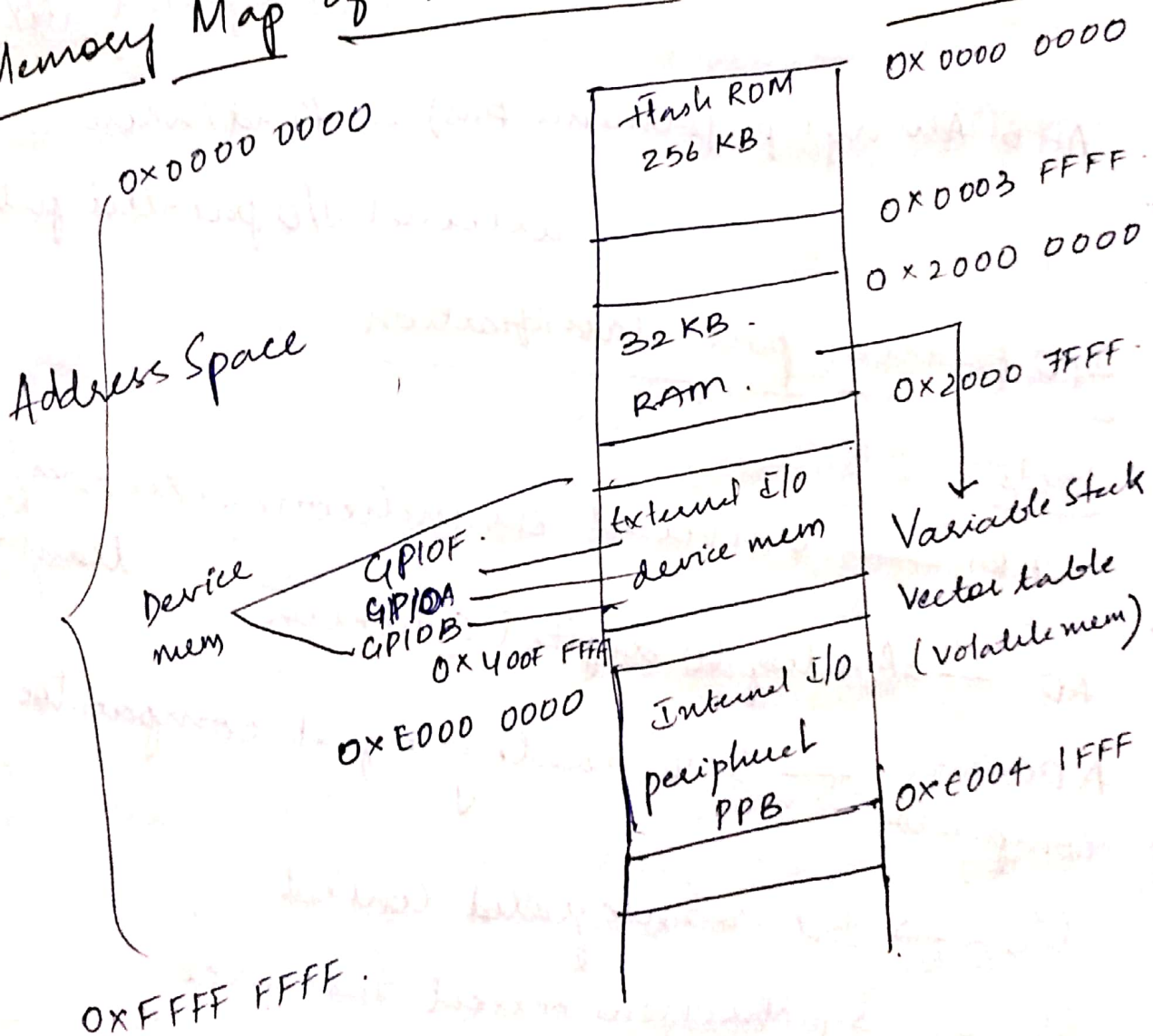
Exception & Interrupt Handling

① Exception Handling (types of exception)
② Interrupts (IRQ, FIQ)
③ Interrupt handling Schemas.

1. Error
2. Interrupt
3. Undefined Opcode

5/02/2017 ??

Memory Map of TM4C123GH6PM Micro Controller



TM4C123 provides diff buses to communicate these banks of mem

Icode bus — fetch the operands from ROM

Dcode bus — Read the const data from ROM

System Bus ⇒ Read/write variables, I/O devices fetch the operand from RAM

PPB (private peripheral Bus) → Read/write internal I/O peripheral (NVIC)

AHB (Adv High performance Bus) → Read/write external I/O parallel ports

I/O parallel port Classification

Ports

UART — Universal Asynchronous receiver/transmitter

ADC — Analog to digital converter

ADC comparator — Two analog signal comparator

I²C → Inter-Integrated Control

SSI — Synchronous serial Interface

QEI — Quadrature encoder interface

PWM → Pulse width modulation

Ethernet — High speed n/w interface.

CAN — Control area n/w.

Timer — period interrupt control.

USB — Universal serial bus.

Fun^n

Port

UART — Useful for communic^n b/w computers (Asynchronous mode of communic^n allow receive/transmit simultaneously)

ADC — Analog sensor interfacing to compare the amplitude of the signal & convert into digital.

ADC comparator — It compare 2 analog signals & whichever is having greater amplitude convert into digital

I^2C — Interface with low speed sensors (Also used for attaching many sensors)

SSI — It also know with a name serial peripheral Interface (SPI) medium speed sensor.

QEI — Useful for attaching BLDC motor

TM4C – ~~I/O H po~~ used to apply the variable power supply of at PWM pins also used to control speed of the motor (Simulate the DAC)

CAN – Used in automobile control sys

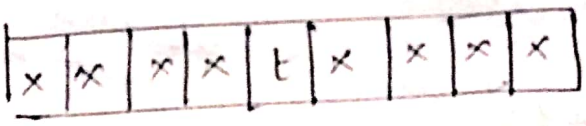
Ethernet – Bridge the internet using the port

USB – Interface with your computer for serial communication.

Timer – periodic interrupt, freq count, PWM counts

Types of Ways to manipulate the reg. bits

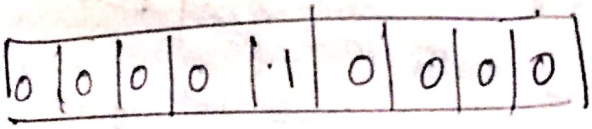
GPIO_Data_Port F

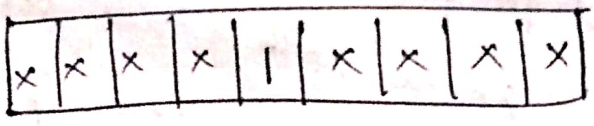


OR → 1

b|0 = b

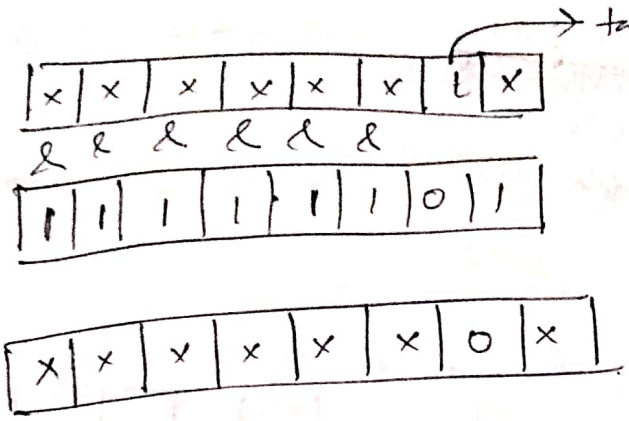
b|1 = 1





GPIO_Data_Port F

= GPIO_Data_Port F | 0x10



add
b|1 = b
b|0

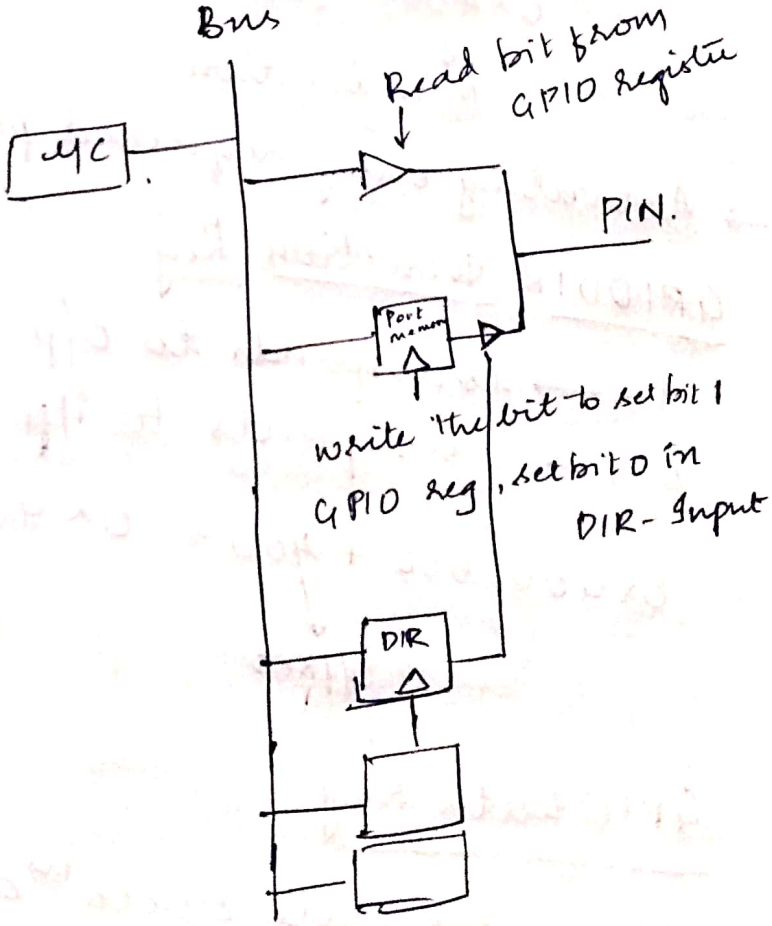
GPIO_Data_PortF = GPIO_Dclk_PortF & 0X02

11/02/2019

IAR Workbench for ARM Microcontroller

GPIO Port

Serial
parallel.
timer

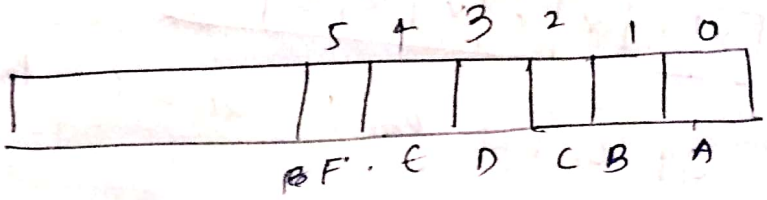


For digital - 4 reg

System clock - can't access port memory
↳ set to active port memory.

Clock gating reg

1 – enable
0 – disable



PortF :- 100000 $=_{16}|32$
2-0

0x20 → Hexadecimal.

0x400FE608 = 0x20
↓
Mem location

→ Activating only required ports saves power.

GPIODIR Direction Reg

1 ⟶ corresponds to o/p
0 ⟶ corresponds to i/p

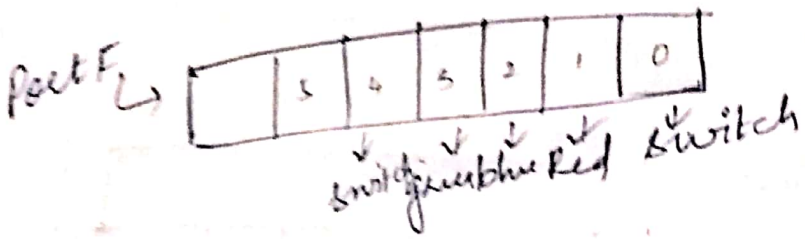
0x40025000 + 400 = 0x40025400
↓ Offset ↓ Mem locn

GPIO Data Reg:

→ All read & write operations are done in this.
→ Individual bit can be selected & manipulated in ARM µC.

> If you need not do take the entire byte to set a particular bit.

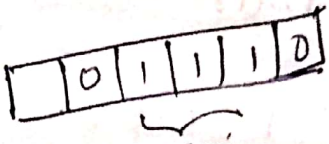
7 6 5 4 3 2 1 0 — pin (8 bit)
9 8 7 6 5 4 3 2 1 — Add (10 bit)



0 0 0 0 1 1 1 0.

⇒ 0x40025000 = 0x0E

```
int main()
{
    *(Unsigned int *(0x400FE608)) = 0x20;   // clk gating
    *(Unsigned int *(0x40025400)) = 0x0E;   // Direct reg
```

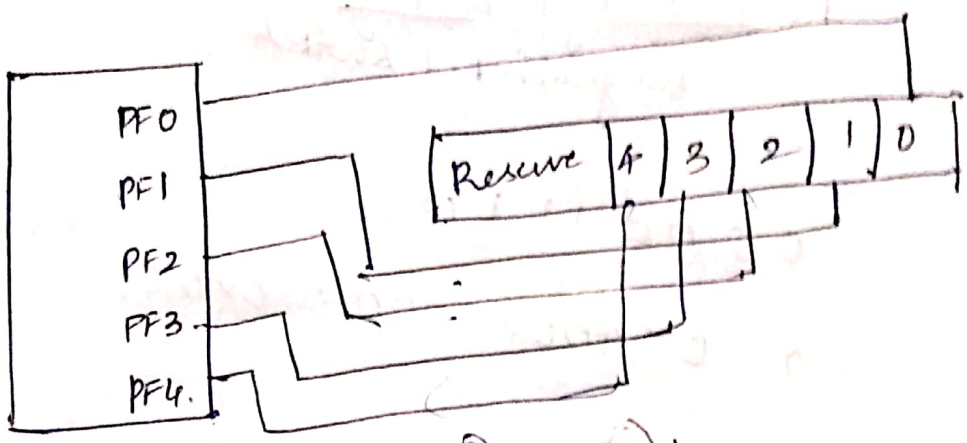



```
    *(Unsigned int *(0x400253FC)) = 0x0E;   // Digital fn.
    *(unsigned int *(0x40025000)) = 0x02;   // to on red LED
    *(Unsigned int *(0x40253FC)) = 0x00;
```

local variable → mem allocated in the reg.

Global " → mem allocated in the RAM.

GPIO Data Reg:-



2 1 8 4 2 1 0 8 4 2 1 0

9 8 7 6 5 4 3 2 1 0

5 4 3 2 1 0

PortF base address — 0x40025000

1st bit → 0x40025008 → 0x02

0x40025010 → 0x04

0x40025020 → 0x08

Pull up reg is write protected

To activate pull up reg

0x400 — = 0xFF

→ PF0 → locked initially

PF4 → Unlocked

To unlock special → 4C4F434B
port

```
*(unsigned int*)0x400FE608) = 0x20U;
*(unsigned int *)(0x40025400) = 0x0EU;
*(unsigned int *)(0x40025510) = 0x0FU;
*(unsigned int *)0x40025510) = 0x11U;
```

10001

```
while(1)
if (*(unsigned int *)(0x40025040) == 0)
    && *(unsigned int *)(0x40025004) == 0
*(unsigned int *)
```

15/02/2019

2 1 8 4 2 1 8 4 → address

8 7 6 5 4 3 2 1

7	6	5	4	3	2	1	0

SW2 SW1

$2^8 = 256$

portF

0x40025000

25 regs

$40025000 + 0x0008 = 0x02$

$40025010 = 0x0F$

'volatile' key word :-

Allows to change values at locns.

locns → Only where there are read/write permissions

GPIODATAF |= (1<<2)

↳ Not atomic

D01

18/02/2019

Internal peripheral device — Used when you want time based systems (timer).

General purpose Timer Module — 16/32 bit GPTM blocks

GPIO Timers operate in diff modes:

→ One shot / periodic mode.
→ Real time clo
→ I/p edge count mode.
→ I/p edge time mode.
→ PWM mode. —— Used to control servo motors (Analog o/p)
→ Wait for trigger mode

- wait for trigger mode:
  Continuously reads sensor reading, whenever the value changes, it acts accordingly (event occurs)
  eg: Watch dog timer.

→ To activate timer circuit,
RCGCTIMER has to be activated
↓
Used to activate GPIO timers.

Exception handling:

Exception state:-

Inactive
Pending
Active

Interrupt vector table.

Reset — priority 1
NMI — " 2
Hard fault — " 3
Memory Mgmt fault — " 4
Bus fault — } Sys generated interrupts

→ IS Accessing when there are no permissions.

→ Interrupt Service Routine handles all user generated interrupts.