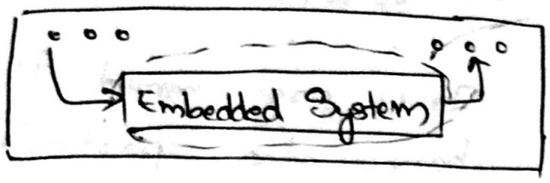
EMBEDDED SYSTEMS

2/1/2019

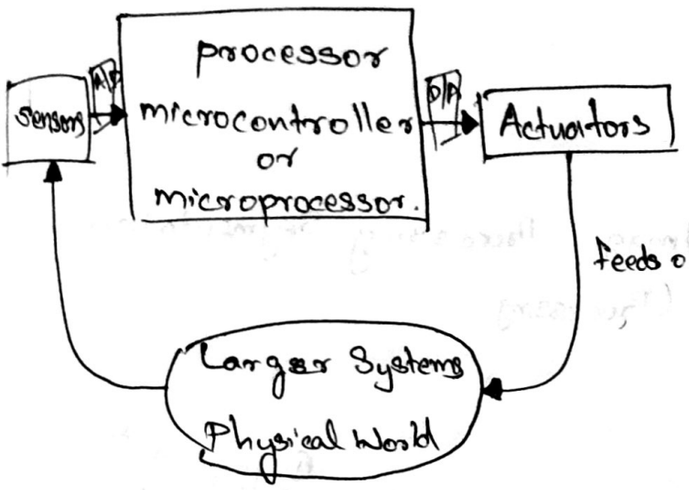
Embedded Systems

Combination of microcontroller/microprocessor which has been embedded to a larger system (physical world things).



→ sphere of control.
i/p → from sensors, controllers.
o/p → Basically actuates something.

Eg:- Washing Machine, Automated Cars, Rockets, Mobile phones.



Provides controlled system to automate things.

A/D → Analog → Digital.

→ Embedded Systems ≡ Cyber physical System ≡ IoT (extends to)
↓ processor ↓ car....

Textbook: Intro to. Arm Cortex-M Microcontroller (2012.)

→

General Purpose System	Embedded System
• Intel Pentium etc..	• Single purpose.
• Multipurpose / Multitasking.	• Tightly Constrained.
	· low cost
	· low power
	· portable
	· Sometime Real time.

Microcontroller → Everything embedded inside the same chip. (RAM, ROM etc..)
limited mem capabilities.

• Microprocessor - Only processing element is there, rest are kept outside

↳ Costlier than microcontroller

→ Real Time System

- Hard RTS. (Conveyer Belt etc.)
- Soft RTS (Camera)

Stringent time is not Required.

Embedded System

- RTS
- No RTS.

Can be programmed to occur to happen at a particular time

•RTC is not possible in ES.

→ ES eg:- Camera.
Single functioned.

→ Reactive vs Transportation → Image Processing Segmentation (Processing

↓

Reacts to event.

(Shutter close & open etc..)

•

Processor

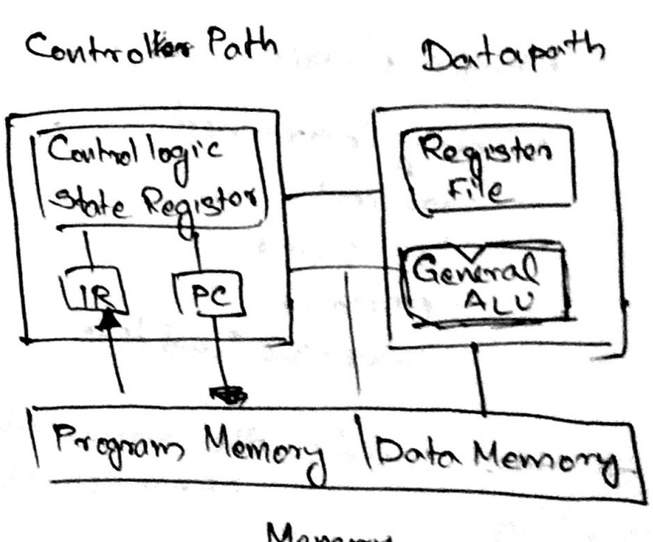
→ General Purpose Processor GPP
(Mobile, Computers.. processor)

→ Application Specific Instruction Processor.
(Microwave, Wash.M.)

→ Application Specific IC. (ASIC)
"Not programmable."

① Differs in fetch, decode, execute cycle
② ASIP can only do

Architecture of Processor



IR → Instruction Register

Memory.

Von-Neuman Single bus Progr. Mem, Data mem. etc...

Hard-ward Multiple busses possible, Pipelining possible

4/1/19

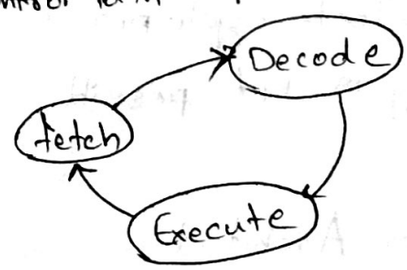
- Soft RTS → Weather Forecasting
- Clock drift in RTS. - Problem

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

Data Path -

Instruction Set Arch. (Simple)

Control Path = FSM → H/W based
→ Microprocessor Program



① → In GPP → Extensive
→ ISet Complex
→ Multitasking is possible.

② ASIP → One application
→ No multitasking.

- In ES, IT is customized, no multitasking.

③ • Single Purpose, No program memory.

↳ Fulfilled by H/W. No logic involved later.

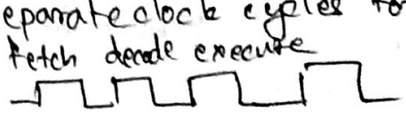
ADC ICs.

SoC.

Architecture

Von-Neumann

- Single memory to be shared by both code & data.
- No pipeling is possible
- Higher speed, less time consuming
- Separate clock cycles for fetch... fetch decode execute



Harvard

- Sep. memories for code & data.
- Pipelining is possible.
- Slower speed, More time consuming.

How Processor Differ from each other

- ISA, Instruction Set Architecture.

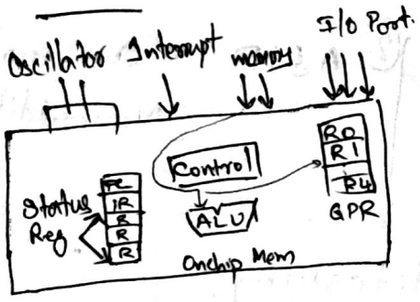
CISC

- Larger set of information. Easy to program
- Simpler design of compiler, considering larger set of instr.
- Many addressing mode causing complex instr. formats
- Instruction length is variable.
- Higher clock cycle per sec.
- Pipelining is not possible

RISC

- Smaller Difficult.
- Complex design for compiler.
- Few addressing modes,

Microprocessor Architecture



Addressing by microprocessor.

Oscillator → Provides sequential clock.

How do we select Microprocessor ?

AC Compressor.
Control unit
Temperature Sensor.

(a) familarity
(b) Speed
(c) Cost
(d) Power
(e) Size

new mc. exists in market

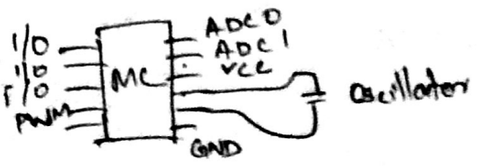
- Arduino - 16 Mhz.
- TNC, ? ARM - 50 Mhz

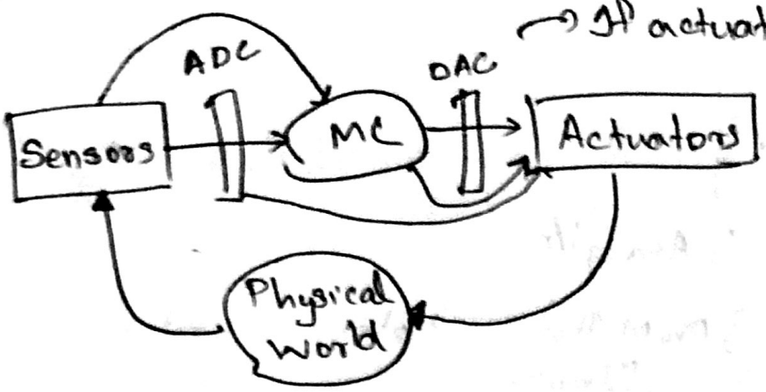
ASIP → Embedded Systems.

Microcontroller.

- Special circuitry, ports will be there

PWM → Pulse Width Modulation



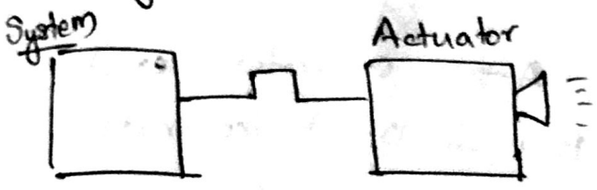


→ If actuator works on Analog and i/p is Digital

↳ Servo Motors
DC Motors etc..
DLC, Stepper
Vibration Motor.

→ Watch Dog Timer

Only present in microcontroller.



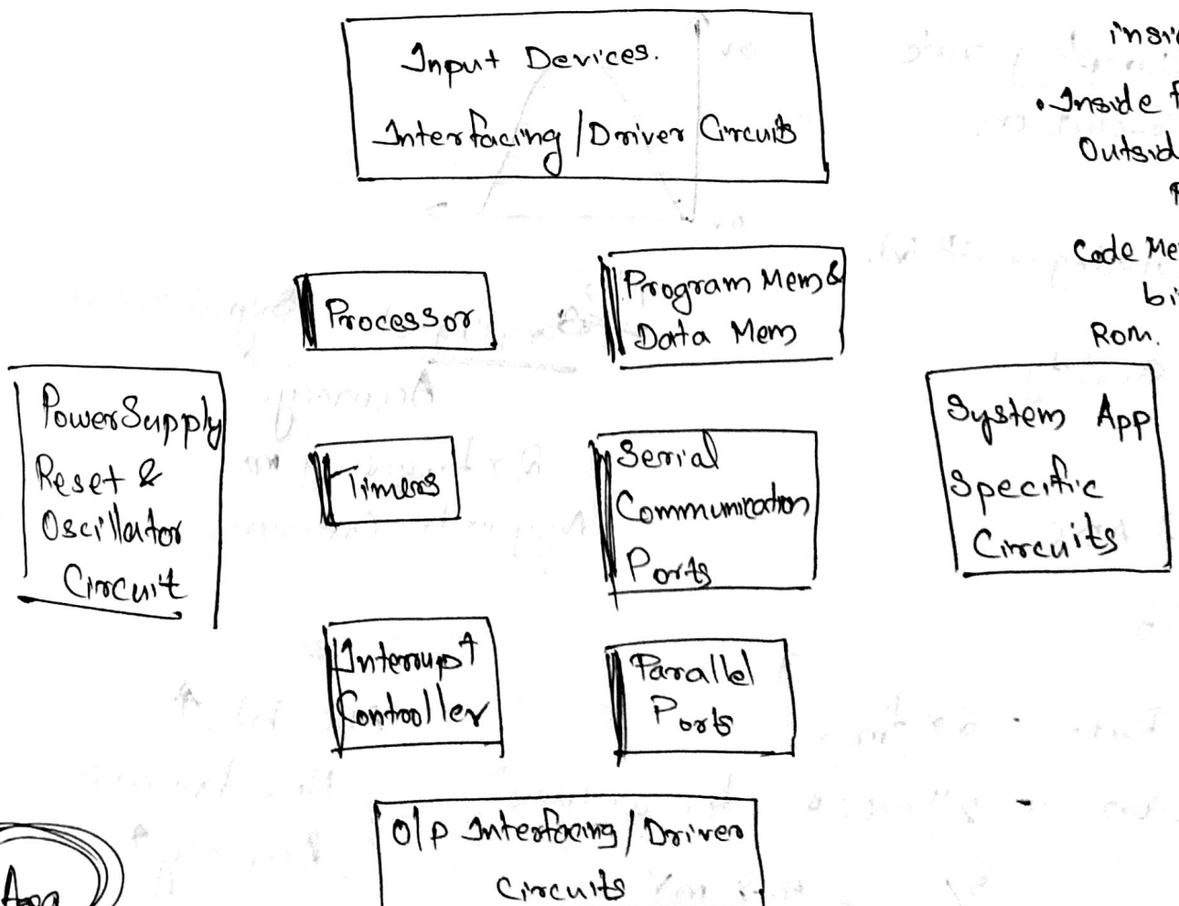
* • event based timer.
• Not only for timing
Used for diff. sensor i/ps as well

→ DSP Digital Signal Processor.

- Discrete Fourier Transform DFT.
- Fast fourier Transform FFT
- Specialized processors for multiplication.
- Multiple Accumulate Instruction.

→ Radar/Satelite communications.

Element of Embedded System

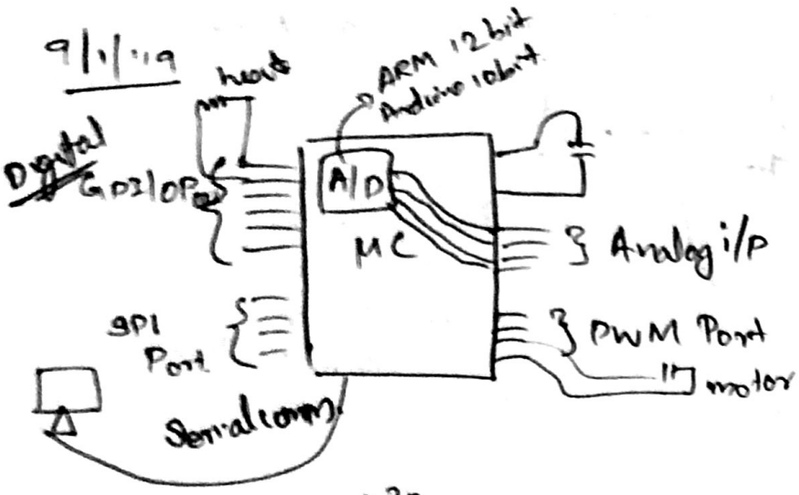


Microcontroller
EPROM resides inside MC.
• Inside Flashdrive.
Outside HDD.
RAM → 4MB
Code Mem, Pocharge bit mem.
Rom.

Analog
Motors
Sensors
SPI Serial peripheral Interface
→ Digital
→ Analog

Ass
What
How it works
what it will give as.

9/1/19



Digital I/O Port - Digital Sensor

Motors, Actuators.

~~If (temp>30)~~

If (temp>30°C)

GPP
DSP vs. GPP Microcontroller A→D converter exist
But no D→A converted

→ DSP → A/D and D/A both are present

Digital Signal A/D D/A

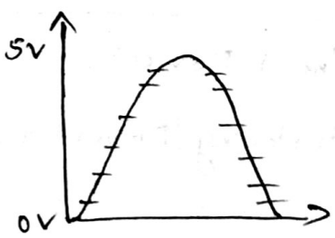
- TMC, → by default, ports are off.

  Register to be used to activate → Clock Gating Register.
  (Save power.)

Analog i/p Port

Attached with analog sensors.

- Sampling rate
- Resolution.



Generally analog → Volt (v).

A→D, Quality

A/D Sampling rate → Signal recovery.
Accuracy.
2 x frequency max
Nyquisite Criterion

Resolution

Bit of ADC.

$2^{10} = 10$

- Sampling Rate = $2 \times f_{max}$.
- Resolution = $2^{10} = 1024$, for arduino.

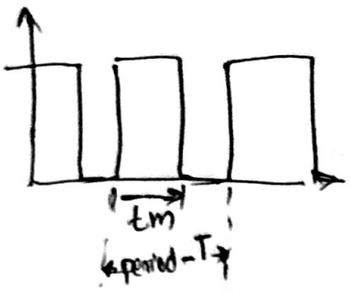
  $= 5/1024 = 4.8$ mV

Smallest measurable unit →

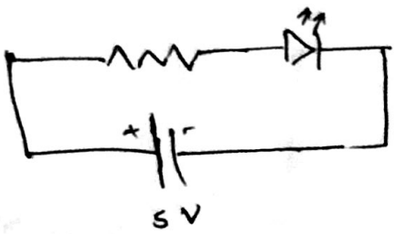
ADC bits ↑
More division possible
Accuracy ↑

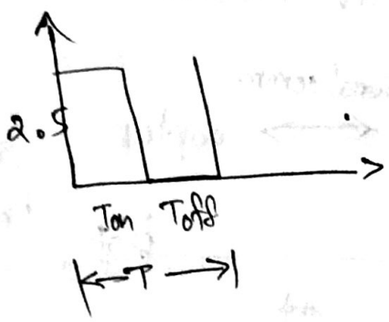
Pulse Width Modulation (PWM)

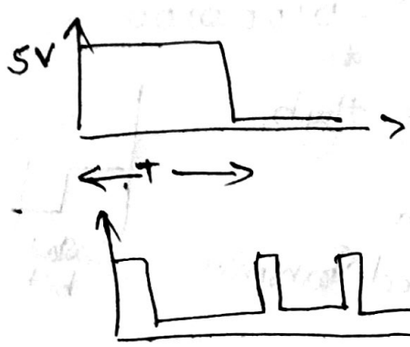
→ PWM is defined in terms of its period & its duty cycle.



Duty cycle: $\Delta = 100 \times \frac{ton}{T}$ (%)







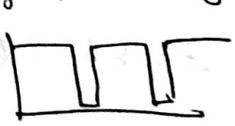
Analog O/P

- PWM is used to simulate the analog o/p.
- PWM is used to control the speed of DC Motors.

$$\text{Output Voltage} = \frac{\text{Pulse on time}}{\text{Total period (on+off)}} * 5\ V$$

$$= 75/100 \times 5 = 3.8$$

For 75% Duty cycle.



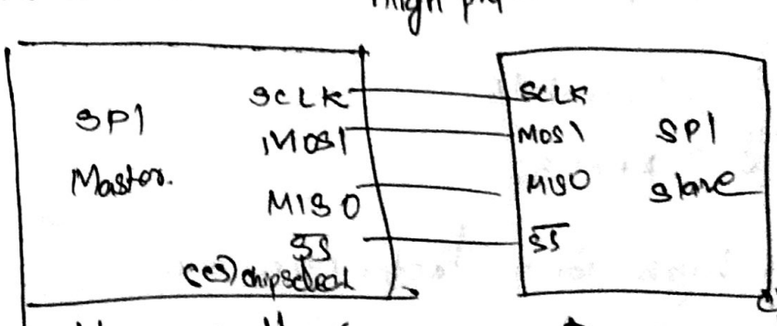
- Speed of Motor
- Brightness of LED

SPI

Serial Peripheral Interface

- Another way of digital input & output is through SPI sensors device.

Align properly



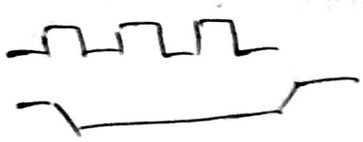
Serial Clk = Serial Clk.

MOSI → Master output slave input

MISO = Master input Slave output

SS/ $\overline{SS}$ → Slave selection

SS low → Transition happens.



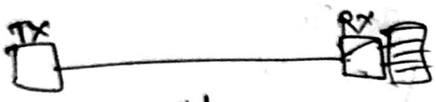
- Wifi Sensors / Module
- Bluetooth Module.

Serial Comm

- Arduino ⟷ Laptop

Serial comm

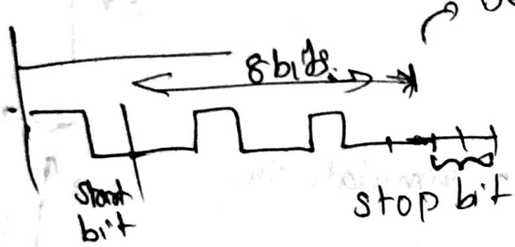
Serial → Parallel.



'a' = 01000100
MSB
LSB

Initially. High.

Depends on Spl. comm.



Synchronation.
Synchronized Transmission

Detects → Using Count.
Here 8 bits.

9/1/2019

Embedded System Design Using ARM

32 bit & 64 bit
- ARM Cortex A (App. Specific)
- ARM Cortex R (Real-Time App)
- ARM Cortex M (General Purpose Microcontroller)

ARM - RISC Architecture.

Arch
- RISC
- CISC.

RISC Compiler → Processor (Simple)

CISC Compiler → Processor Greater Complexity.

High Code Density
- Thumb 16
- ARM 32.

Low cost and high form factor (Die area)

Debugger Technology (JTAG)

ARM Design Philosophy

High Code Density.
Low Price & small form factor.
Debug technology - JTAG
Advanced RISC m/c not pure RISC → some will take more cycles.

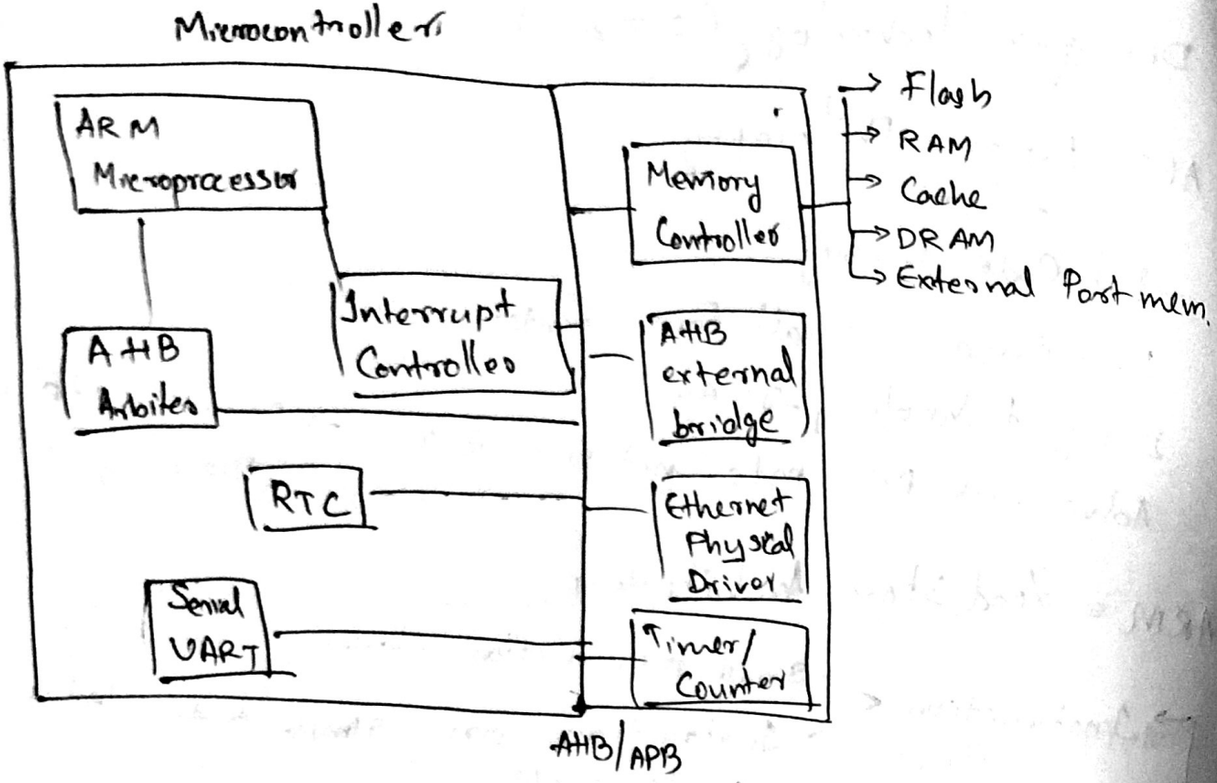
ARM = Load Store Architecture

- Instruction ← Small # instruction classes.
  Instructions are simple & even led in single cycle.
- Pipeline. → Parallel execution

- Register → memory. General Purpose register.
- Load/Store Arch.

not Pure RISC Arch.
10 Instruction — Variable cycle Executes for certain instruction
Load/Store - Multiple

- Inline Barrel Shifter.
- Thumb Instruction - 16bit (High Code density)
- Conditional Execution.
  (Add if Carry/Overflow/Zero)
- Enhanced Instruction (-) faster execution.



AMBA architecture: (On chip bus architecture)

↳ Advanced Microcontroller Bus Architecture.

↳ 1996.

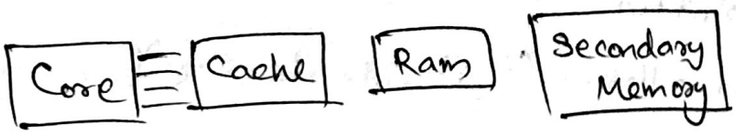
↳ APB (Advance Peripheral Bus)

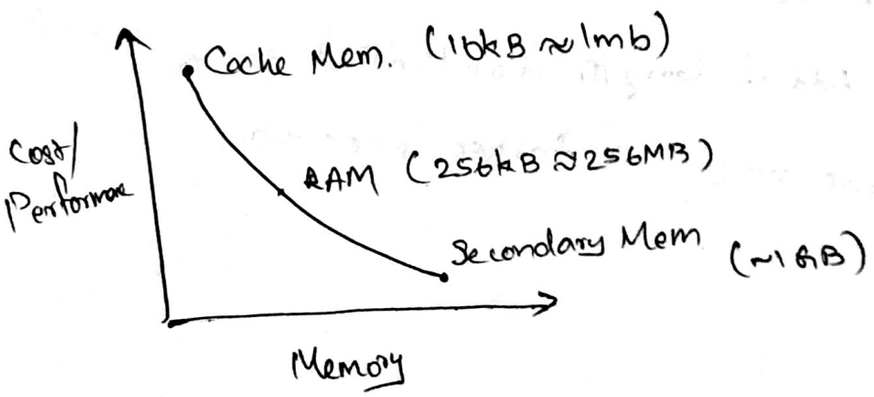
AHB (Advanced High Performance Bus)

vs

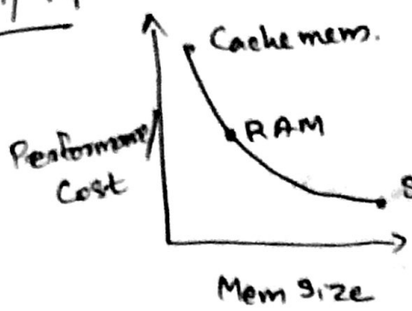
→ PCI controller. (external bus architecture) → Graphic Card
Wifi card

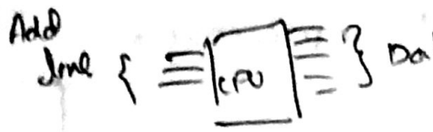
ARM Memory Design





14/1/19



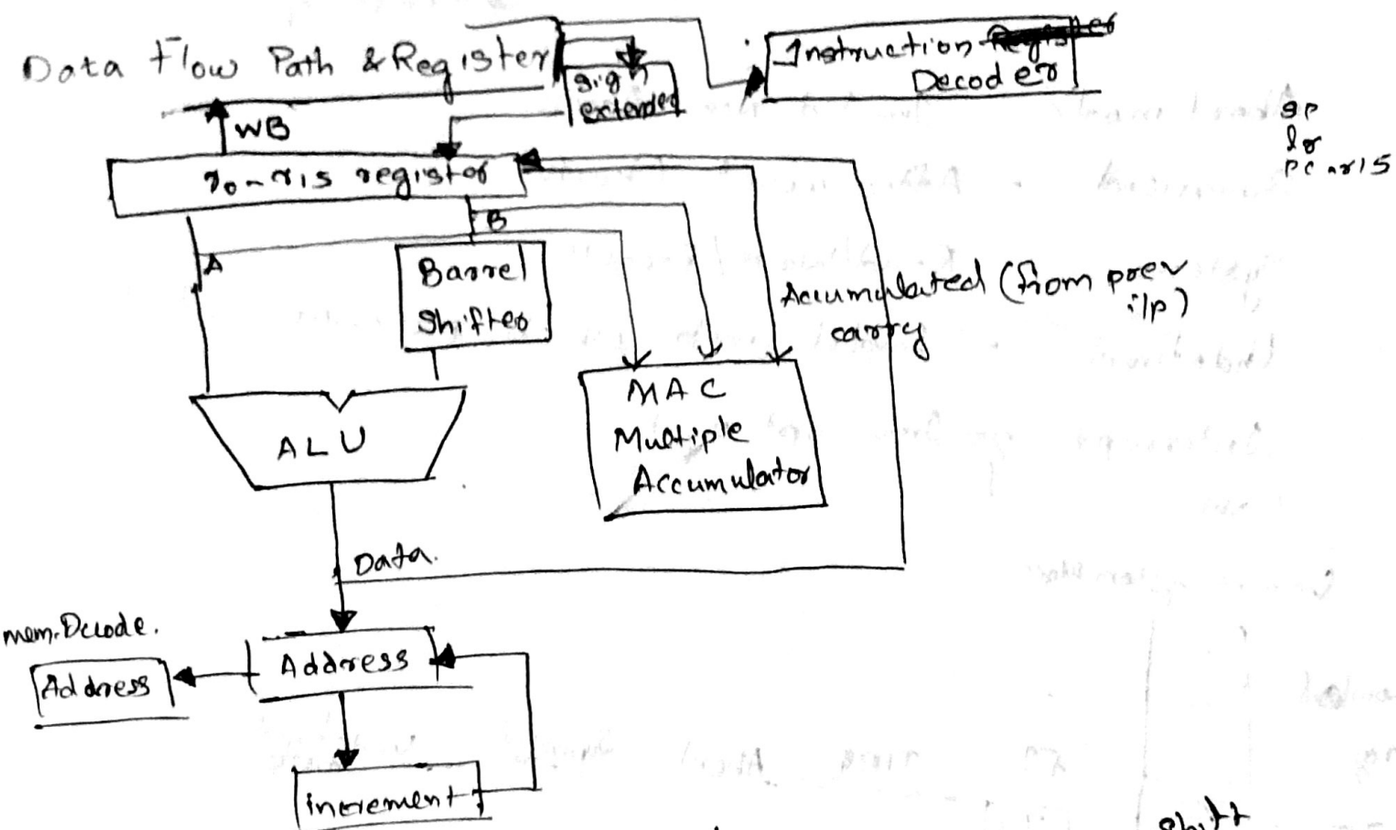


Instruction size
ARM : 32 bit

Data width
ARM : 32 bit
Memory Size : 16 bit (2 byte)

Instr size	8-bit Mem	16 bit mem	32 bit mem
ARM 32 bit	4 cycle.	2 cyle	1 cycle.
Thumb 16 bit	2 cycle	1 cycle	1 cycle.

Data Flow Path & Register



sp
lr
PC - r15

- Sign extender 16 bit → 32 bit
- Barrel shifter : Allows logical opns. &&, ||, <<, >> Shift
- Register Bank : 18 register inside the ARM microprocessor that is visible to a programmer.

- All 18 set of registers

r0 ⋮ r12 } GPR / Orthogonal Register

Special fn Register
- r13 sp (stack ptr)
- r14 lr (link register)
- r15 PC (pgm ctr)

CPSR - Current pgm status register
SPSR - Saved pgm status register

What is mode?

ARM Microprocessor - 7 modes.

- Non-privileged mode (User mode)
- Privileged mode
  - Abort
  - Supervised
  - System mode
  - Undefined mode
  - Fast Interrupt rqst
  - Interrupt rqst

Abort mode - Invalid mem Access.

Supervised - After reset (kernel.

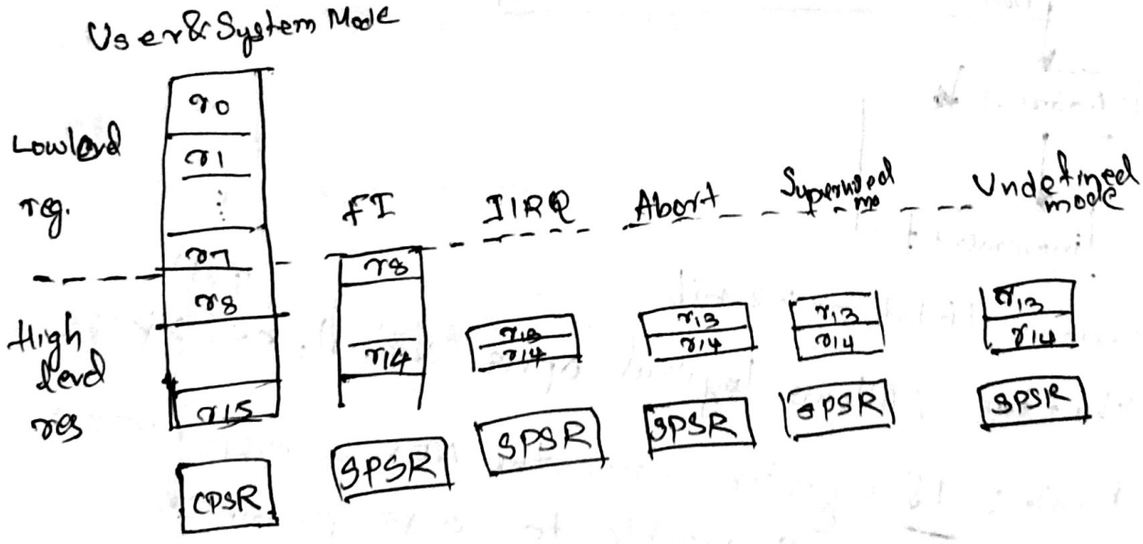
System - Read/Write/Execute

Undefined - Invalid instr. which processor can't decode

Interrupt } - Interrupt rqst.

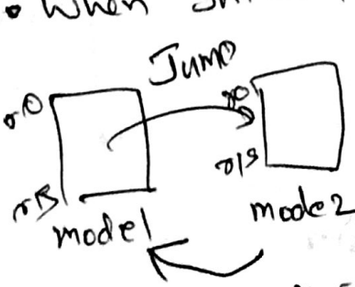
Fast

User & System Mode



Total 37 register.

- When Interrupt happens Interrupt handler



SP → Saved the info from r0-r12

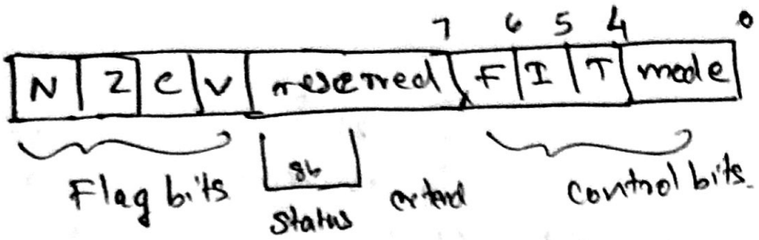
SPSR - Processor status.

Reverse → CPSR

Program Status Register

CPSR
SPSR
} 32 bit

conditional execution



N - Negative
Z - Zero
C - Carry
V - oVerflow

F - Fast Interrupt
I → Interrupt Request
T - ARM/Thumb.
mode.

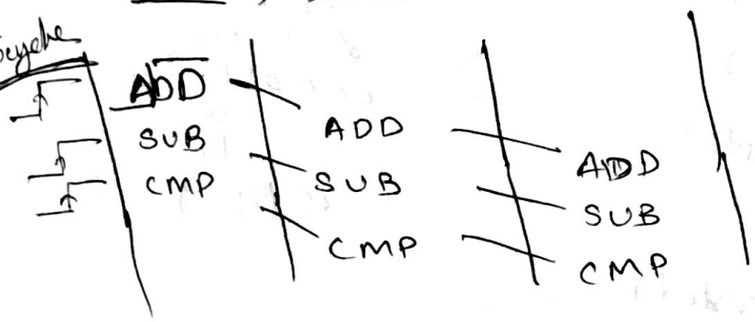
25/1/19

Pipeling Architecture of ARM Microcontroller Fam

Arm7

Fetch → Decode → Execute

ADD - 3
SUB - 3
CMP - 3
9 Cycle.



ARM10

Fetch → Issue → Decode → Execute → mem. → write.

Issues to next inst flag

latency

ARM9

Fetch → Decode → Execute → Mem → Write.

Instruction set of ARM microcontroller

- Data processing instn
- load/store instn
- Pgm status register instn
- Software interrupt instn

- loading constant instn
- Branch instns
- conditional execution instn.

① Data processing instn

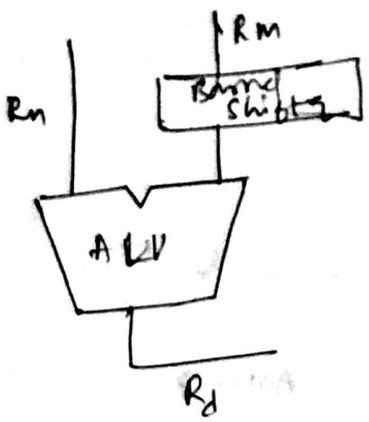
mov {S}

Syntax

<instn> <condn> <S> <R_d> <N>

- <instn>: mnemonics
- <condn>: optional
- <R_d>: destination register
- <N>: {0-31} ↑ int const

r0=5
r1=7
mov r0 r1;
r0=7
r1=7



Barrel shifter preprocessing command

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

mov r5 r7 LSL#2

00000111
11111101
00001110
↓
14
r5=14 r7=7

CPSR — N Z C V I F T → user mode

N=1 Z=1 C=1

condn - Add if carry occurs
Sub if
ADD if zero flag is high

r7=00000000
r5=10001000
mov r7 r5 LSL #1
10001000
(1) carry occurs

28/1/19

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

Arm Instruction
- Thumb Instruction (16 bit)
  - can't support conditional execution along with dataprocessing Inst.
- ARM Instruction (32 bit)

Syntax

Thumb <Instruction><S><Rd><N> (0-31 bit)

ADDEQ X

ARM <Instruction><cond><S><Rd><N>

ADDEQ ✓

- ARM instruction takes larger space. So using Thumb instruction increases code density. More # instructions.

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

GCD. (Greatest Common Devisor)

Thumb (16 bit)

7 x 2 byte = 14 bytes

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

ARM (32 bit)

4 x 4 byte = 16 bytes

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

Arithmetic Instructions

<instruction> <cond> <S> <Rd> <Rn> <Rm>

- <Rd> → destination register.
- <Rn> → Source Register.

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 Rd, Rn, N; ⇒ Rd = N − Rn;

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

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

r0 = 0x 0000 0001;

SUB r0, r0, #1; r0 = r0 − 1

r0 = r0 − 1

0x 0000 0001

0x 1111 1110 +

0x 1111 1111

SUBS → updates C & Z.

Logical Instruction

↳ 3 PSR.

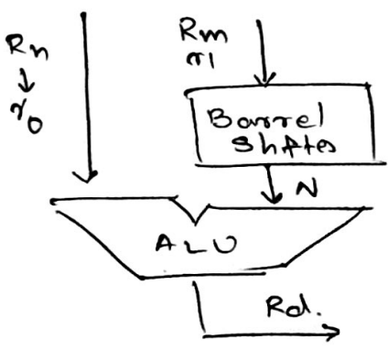
AND Rd, Rn, N; ⇒ Rd = Rn & N

ORR Rd, Rn, N; → Rd = Rn | N

EOR Rd, Rn, N; → Rd = Rn ^ N

BIC → Bit Clear

Rd Rn N → Rd = Rn & (~N)



PWM - Motor.

4 categories of sensors.

SUBS ↳ updates C & Z.

Logical Instruction

↳ CPSR

AND Rd, Rn, N; ⇒ Rd = Rn & N

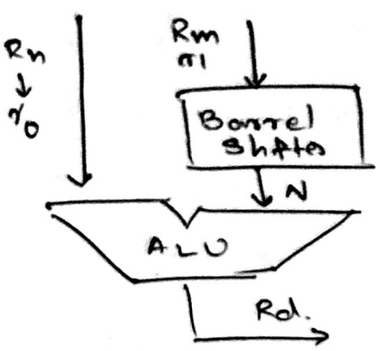
ORR Rd, Rn, N; → Rd = Rn | N

EOR Rd, Rn, N; → Rd = Rn ^ N

BIC → Bit clear

Rd Rn N → Rd = Rn & (~N)

PWM - Motor.



4 categories of sensors.

29/1

COMPARISON INSTRUCTION

CMP → Comparison

CMN → Comparison negated.

TEQ → Test the Content of two 32 bit register value.

TST → Test the bit of 32-bit value.

Syntax

<instruction> <cond> <Rn> <Rm>

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

Arithmetic →

CMP r0, r1; ⇒ r0 − r1 ↳ update flag

CMN r0, r1; (r0 + r1) ↳ update flag.

TEQ r0, r1; r0 ^ r1

TST r0, r1; r0 & r1

CPSR

N	Z	C	V

Multiply Instruction in ARM Microcontroller

(convolution used in DSP)

Syntax

MLA {cond} {S} Rn, Rm, Rs, N ⇒ Rn = (Rm * Rs) + N

MUL {cond} {S} Rd, Rn, Rm; Rd = (Rn * Rm)

Signed & Unsigned

SMLAL (Signed Multiply Accumulate Long.
SMULL
UMLAL
UMULL

r_0 = 32 bit
r_1 = 32 bit
r_2 = 64 bit

UMULL Rd, R0, Rs, N

[Rd, R0] = (Rs * N)

higher bit 32bit

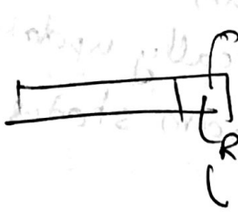
Branch Instruction in ARM Micro Controller

B {cond} label {Branch.} → B forward ⇒

BL {cond} label

BX {cond} label

BLX {cond} label Rm.



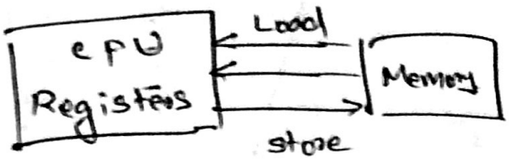
BX

BLX

```
    ADD R0, R1, #2;
    ADD R2, R3, #4;
forward.
    SUB R0, R, #1;
backward
    ADD R4, R5, #1;
    SUB R6, R7, #4;
B backward
```

1/2/2019

Load / store Instruction in ARM microcontroller



- Syntax

Block of Mem Access.

4 byte word aligned

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

STR {condn} B | H | , Rn, Addressing mode.

Addressing Mode (for single register of data)

(i) Pre index with write back.

(ii) Pre index

(iii) Post index

Single register LDR | STR instruction

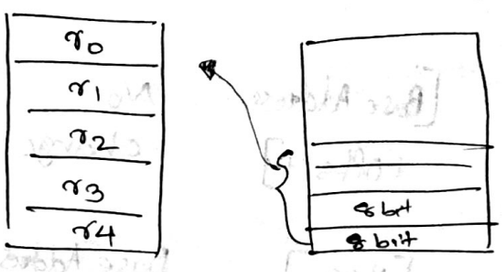
- LDR, STR

↓

4 byte at a time

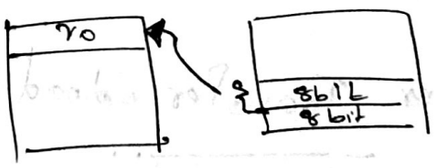
32 bits.

word aligned mem transfer



0x00080010.

- LDRB, STRB.



one word.

B → 8 bit)

H → Half word.

(16 bit)

W → Word

(32 bit)

- LDRH, STRH

2 bytes

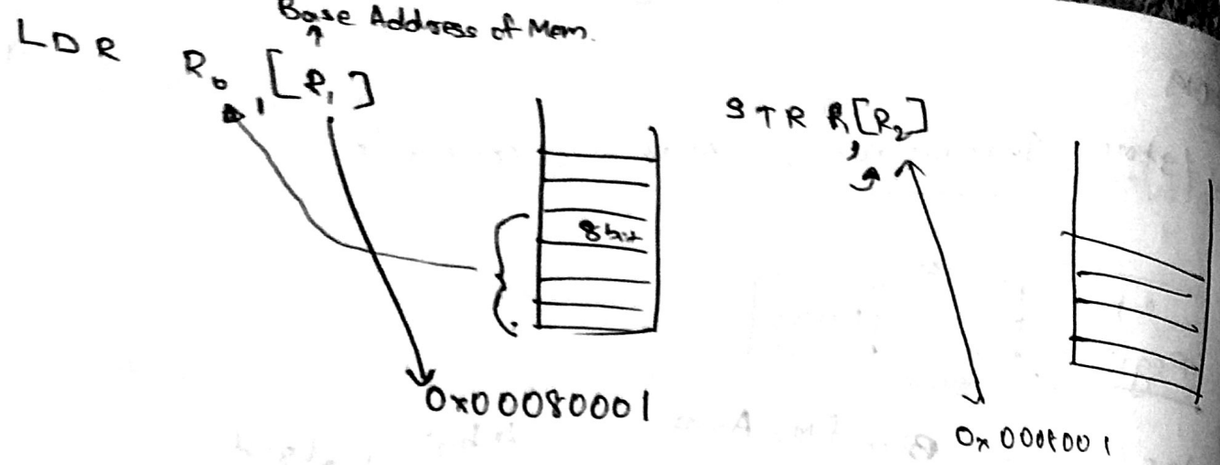
- LDRSB

↓

signed 8 bit mem trans

LDRSH

↳ both not available for STR



(Big Endian LSB → MSB) | Little endian

Indexing Mode.

	Data	Base Address	Example
Pre index with WB mode.	{Base Address + offset.	Baseaddress to offset	LDR R0[R1, #4]!
Pre index	[Base Address +offset]	No change	LDR R0 [R1, #4]
Post index	[Base]	Base Address +offset.	LDR R0 [R1], #4

Multiple Register Transfer load/store Instruction ...

LDM
STM

Syntax:

<LDM/STM> {condition} {Addressingmode} Rn {!} <Registers>

IA (Increment after
IB
DB - Decrement befor.
DA

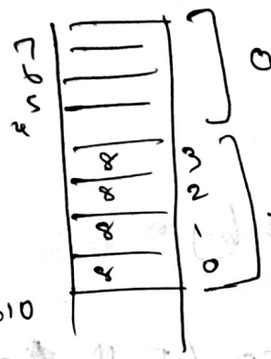
	Start Address	End Address
IA	R_n	$R_n + N*4 - 4$
IB	$R_n + 4$	$R_n + N*4$
DA	$R_n - N*4 + 4$	R_n
DB	$R_n - N*4$	$R_n - 4$

N - Block of Mem Access.

r_0 = 0x0008 0010
r_1 = 0x0000 0000
r_2 = 0x0000 0000
r_3 = 0x0000 0000

Pointing to Base Address of mem.

LDMIA $R_0\{!\}$ $[r_1 - r_3]$



Datas 0x0008 0010

IA
$r_1 = (r_0)$ ← 0x01
$r_2 = (r_0+4)$ ← 0x02
$r_3 = (r_0+8)$ ← 0x03

IB
$r_1 = (r_0+4)$ 0x02
$r_2 = (r_0+8)$ 0x03
$r_3 = (r_0+12)$ 0x04

MUL

(32 bit x 32 bit → 64 bit → stored in register

STMIB $r_1\{!\}$ $[r_3 - r_6]$

R_n r_3 = 0x0001 0001
r_4 = 0x0010 0100
r_5 = 0x0100 0110
r_6 = 0x1000 1000

4/1/2019

Stack Operations in Microcontroller (Part of Memory RAM)

Load/Store Multiple, register instructions is used to ~~implement~~ used to implement ~~stack operations.~~ stack operations.

push → Store data in memory/stack STM{ }

pop → Get the data from memory/stack. LDM{ }

↑ Addressing mode.

STMFD sp{!} <r1, r2>

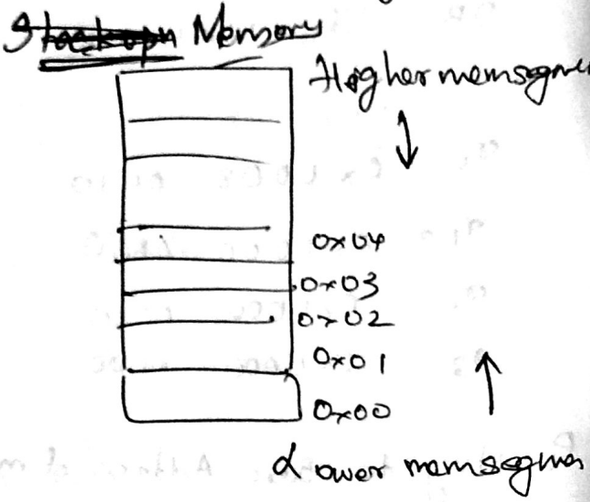
↓

Full descendy

r1 = 0x 0000 0002

r2 = 0x0000 0001

sp = 0x0000 0014



sp points

LDM/STM → 4 bytes

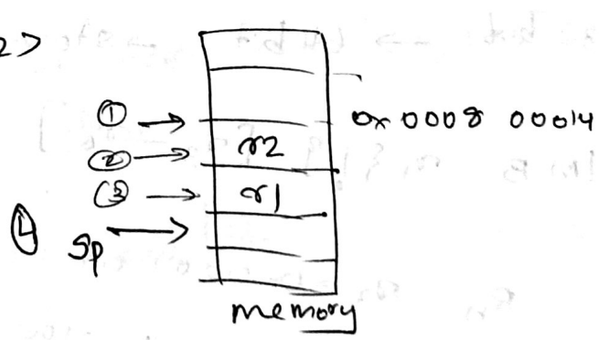
LDM/
STM. FD (Full descending

FA (Full Ascending)

ED (Empty Descending)

EA (Empty Ascending)

STMED sp{!} <r1, r2>



How the Stack can be utilized

ATPCS

↳ ARM-THUMB Procedure Call Standard.

① When Processor change the state.

② Interrupt Occurs.

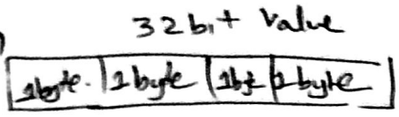
③ Function Call

④ Process Save Current Program status

Big Endian vs Little Endian

→ Memory alignment. Diff → Time to access is

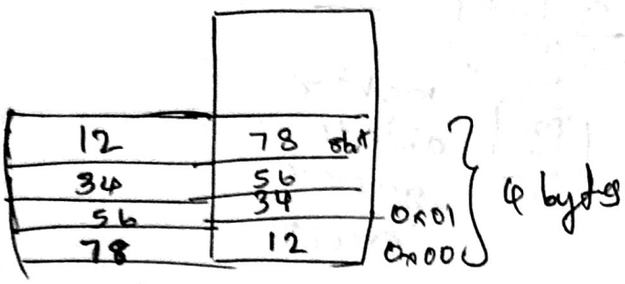
Little Endian



MSB.

12 34 56 78

LSB

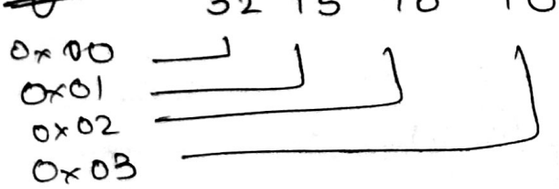


Little Endn

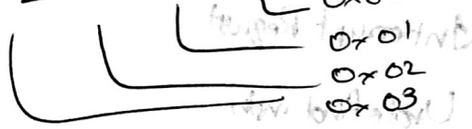
Little Endian → LSB → stored at least address.

int a = 32 15 70 90.

Big Endian



Little Endian



Q) a = 32 15 70 90 b = 15 70 90;

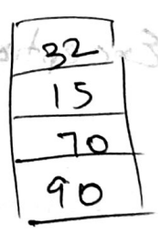
A) for Big endian, we have to shift → RIS C

But for little Endian it is easy to store.

↓

Motorola, Intel, Amn.

ARM follows both.



LE

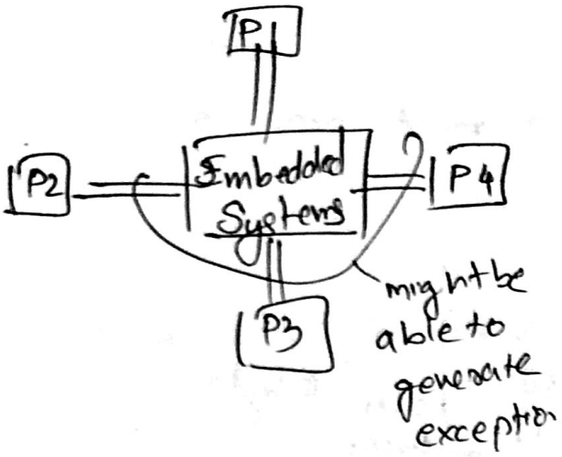
BE

Exception And Interrupt Handling

(1) Exception Hdling (Types of Exception)
(2) Interrupt Hdling (IRQ, FIQ)
(3) Interrupt Hdling Systems.

Performance of Microcontroller depends on good Exception Handler.

(i) Error
(ii) Interrupt
(iii) Undefined operation



5/2/19

Exceptions	Address	
Data Abort	0x0000 0000	Exception Hdlrs.
Prefetch-Abort	04	
S/w Interrupt	08	
Fast Interrupt Rq	0C	
Interrupt Request	16	
Undefind instn	20	
Rese	24	

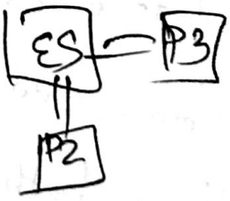
(Common in every microcontroller. i.e ARM.

Basic task initialize each & every register to its previous value.

Void data Abort Exception Hdlr

{

}



Vector Table

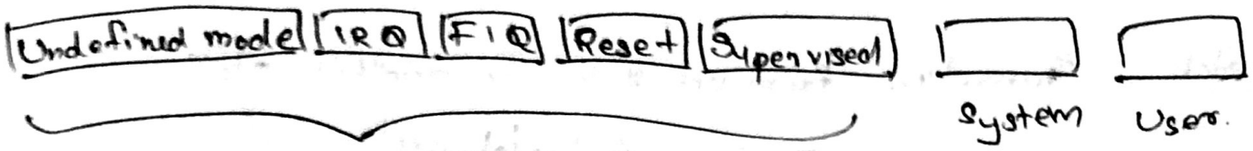
→ When an int Exception occurs, process does these steps.

① Save CPSR → SPSR of exception mode.

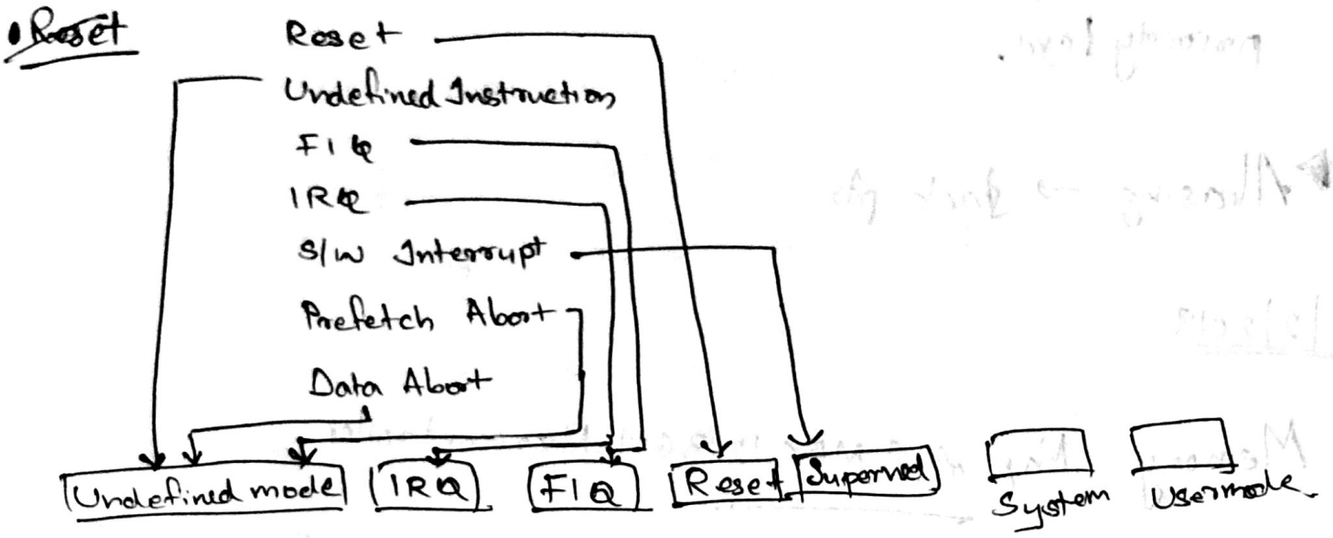
② Save PC → LR of Exception mode.

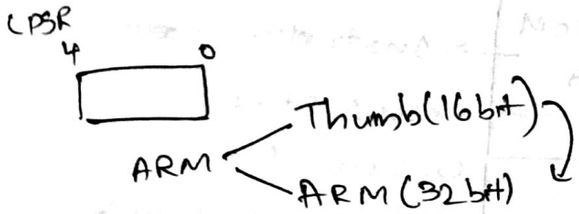
③ Set CSSR register with exception mode, an exception

(iii) Set PC assigned with vector address.



Enters into one of these, when an exception occurs





Exception Handle based on Priority level.

Exception

Reset → 1
Data abort → 2
FIQ → 3
IRQ → 4
Prefetch → 5
S/w Interrupt → 6
Undefined Inst → 6

Implemented by programmer.

Both won't occur at same time. orthogonal.

Reset

clears registers etc .. when 1 pgm is in ME, if you want to upload another pgm

Data Abort

Access Data out of memory

FIQ/ IRQ — Sensors / Peripherals.

Prefetch — Transition from one state to another state

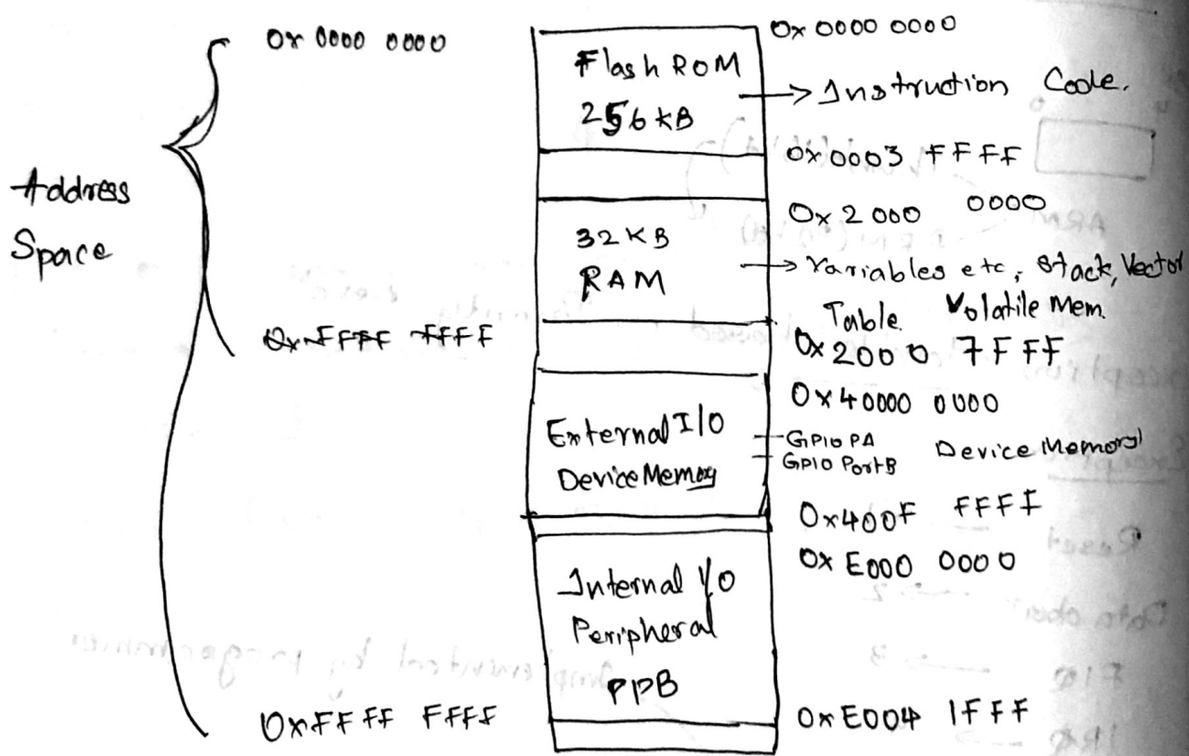
Pipeline states/stages

- When multiple exception occures, handled according to priority level.

→ Aliasing → stack ptr.

8/2/2019

Memory Map of TM4C123GH Microcontroller



Internal
- Peripherals → Timer etc..

→ TM4C123 provide buses to communicate these banks of memory.

I code Bus ⇒ Fetch the operands from ROM

Decode Bus ⇒ Read the constant data from ROM

System Bus ⇒ Read/write Vamables, I/O Devices, fetch the operand from RAM.

PPB (Private Peripheral Bus)

⇒ Read/Write internal I/O Peripheral.

NVIC
Nested Vector
Interrupt Control

AHB (Advance High Performance Bus)

⇒ Read/Write External I/O Parallel pins.

TM4C Ports

UART – Universal Asynchronous Receiver/Transmitter

ADC – Analog to Digital Converter

ADC Comparator → Two analog signal comparator.

I^2C → Inter-integrated central.

SSI → Synchronous Serial Interface.

QEI → Quadratic Encode Interface

PWM → Pulse Width Modulation

Ethernet → High Speed N/W Interface.

CAN → Central Area N/W.

Timer → Period Interrupt Central (NVIC)

USB → Universal Serial Bus

Port	Function
① UART →	Useful for communication b/n computer. (Asynchronous mode of communication allow receive/transmit simultaneously)
② ADC →	Analog Sensor Interfacing to compare the Amplitude of the signal and convert into ~~signal~~ digital.
③ ADC Comparator →	It compare the Analog sensor and whichever is having greater amplitude. convert into digital.
④ I^2C →	Interface with low speed sensors. (Also used for attaching many sensors).

SSI → 2 microcontrollers, using SPI or SPI sensor. It is also known as SPI, Serial Peripheral Interface. (Medium Speed Sensor)

QEI → useful for attaching BLDC Motor.

PWM → Used to apply the variable power supply at PWM ports also used to control speed of the motor (Simulate DAC). DC, Servo Motor, Stepper Motor

CAN → Used in Automobile Control System

Ethernet → Bridge the internet using the port.

USB → Interface with your Computer for Serial Communication.

Timer → To generate periodic interrupts.

Types Of Ways to Manipulate the Register Bits

OR to set

x	x	x	x	t	x	x	x

0	0	0	0	1	0	0	0

x	x	x	x	1	x	x	x

→ 0x08.

GPIO_Data_PORT F

OR → 1

b|0 = b

b|1 = b.

GPIO_Data_PortF | 0x08

AND To Clear

x	x	x	x	x	x	t	x

& & &

Mask | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |

add b

0xFD = ~0x02

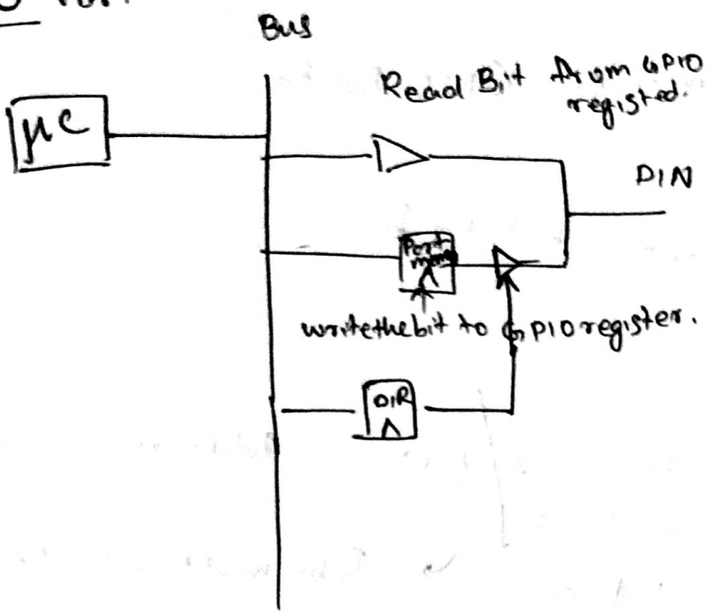
GPIO_PortF = GPIO_Data_PortF & ~0x02

11/2/2019

IAR, ~~IDE~~ Workbench for ARM Microcontroller

↳ Register changes are visible.

GPIO Port



Input / Output.
not both

Digital Opn → 4 opns

Register

Set bit 1 in DIR → o/p

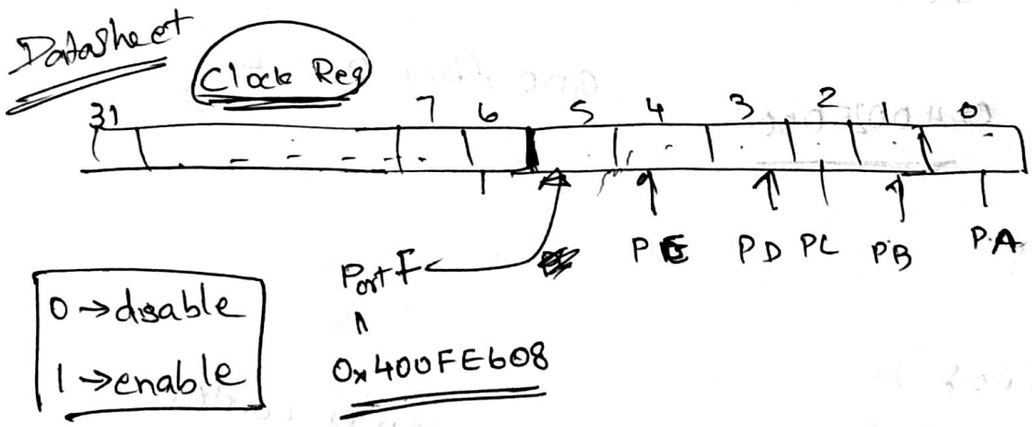
Set bit 0 in DIR → i/p

7 steps for Port Initialization.

① System Clock required to access mem for a clock.

Base + offset → Port Address. 0x400FE608 → Port F in clock

Datasheet



0x400FE608 → 0x0000 0000

↓

0x0000 0020

PF Activation.

Turns mem on.

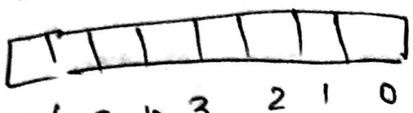
To save Power, Ports are deactivated.

When we want to use, we can activate & use.

Port F. 0x40025000 + 0x51C

Base.

↳ Dir Reg.



GPIODATA

→ Read/write ops performed on GPIO Data.

Aligned with mem lines

Here we can select/set bit by bit.

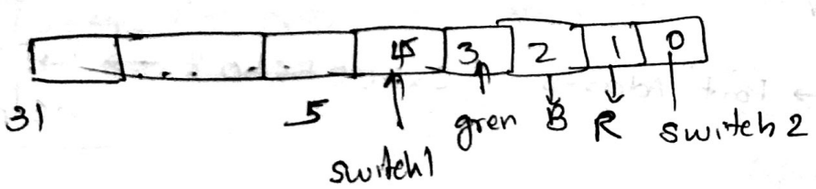
9	8	7	6	5	4	3	2	1	0
-	6	5	4	3	2	1	0		
0	0	0	0	1	1	1	0		

Address Line

↳ To be multiple of 2

0x 0E

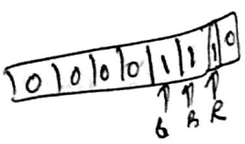
Port F



Port F base add = 0x40025000

GPIO offset 0x524.

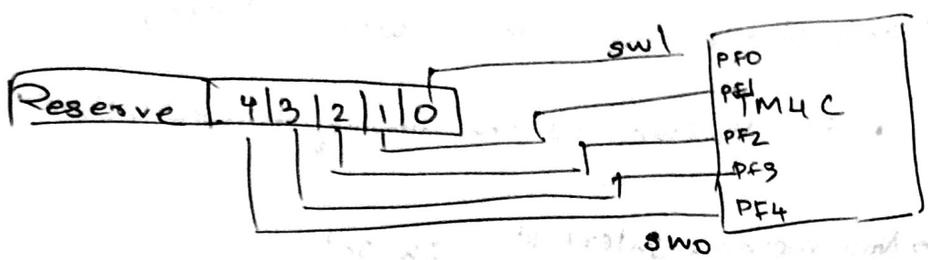
```
int main()
{
(0x400FE608.)
*(unsigned int *(0x400FE608))   → gives content
                                → Clock Gate Registe
Clk Gate  *(unsigned int *(0x400FE608)) = 0x20;
DIR       *(unsigned int *(0x40025400)) = 0x0E;
Digital En *(unsigned int *(0x400253FC)) = 0x0E;
Data      *(unsigned int *(0x40025000)) = 0x02;   //blink Led.
```



local variable → Register Part Allocation

Global Variable → RAM

12/2/18

GPIOF : 

2^7 8 4 2 1 8 4 2 1

9 8 7 6 5 4 3 2 1 0

7 6 5 4 3 2 1 0

or 0x40025008

7th bit - 0x40025200

ARM

256 32 I/O.

Port F 0x40025000.

For all bits. → 0x400253FC

All address in ARM mem are unsigned.

U → represents unsigned.

☆ PF4 is not locked.

pin individual

For Addr line weighted code

9 8 7 6 5 4 3 2 1 0

2 1 8 4 2 1 8 4 2 1

7 6 5 4 3 2 1 0.

~~Blue LED.~~

~~int main()~~

Lock 0x40025520 = 0x4C4F434B.

```
int main()
{
  *(unsigned int *) 0x400FE608 = 0x20U;
                    25400 = 0x0EU
                    2551C = 0x1FU;
                    25524 = 0xFF;
                    25510 = 0x10;   → 1: for switch U
```

)

18/2/19

GPIO → Internal Peripheral

↳ General-Purpose Timer Module (GPTM)

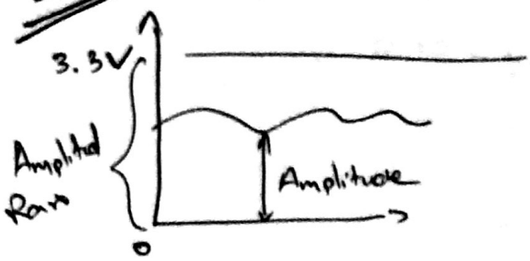
* GPT Module is one timing resource other Systick & Pwm

NMI → Non masking Interrupt - s/w Inter.

↳ 2nd highest prio.

Analog o/p

22/2



Amplitude

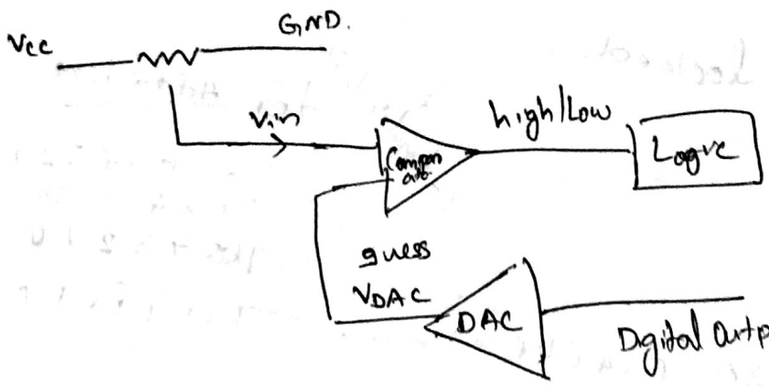
Amplitude Rage.

Time Quantization

Time Period.

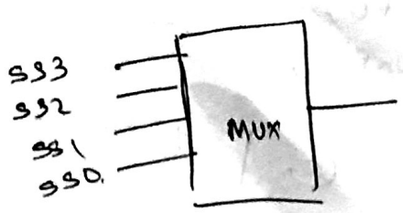
12bit : 4096 → Time.

$= \frac{3.3}{4096} = 0.8mV$



$V_{DAC} > V_m$ → bit set to be low.

$V_{DAC} < V_m$ → bit set to be high.



ADC Interrupt

ADC Interrupt Handler pos = 51.

↳ Set to that