N. vamsi krishna
20KP1A04H81

UNIT - V

ARM ARCHITECTURE AND PROCESSOR

ARM processor: (Advanced RISC Machine)

Used in Music players, Smartphones, wearables, tablets and other Consumable Electronic devices

1. It needs very few instructions and transistors
2. It is very Small size, which is perfect for Small size devices.
3. It has less power Consumption along with reduced Complexity in the circuit
4. It Can be Used for 32-Bit devices & Embedded Systems.

Features of ARM processor: (32 bit processor)

1. Multiprocessing System: ARM processors are designed where more than One processor are Used to process information. First ARM Processor introduced by name of ARMv6k had Capability to Support "4 CPU's" along with its hardware.

2. Tightly Coupled Memory: Memory of ARM processor is tightly coupled this has very fast response time. It has low latency that Can also be Used in Cases of Cache Memory [latency - the instruction starts Executing to get Correct o/p

3. Memory Management: It includes Memory Management Unit and Memory protection Unit.

4. Thumb-2 Technology: Thumb-2 instruction set Combining both

16 bit & 32 bit instructions to increase the performance in new Embedd Systems and Mobile phones with longer Battery life.

5. One cycle Execution time: Each instruction is of fixed length that allows time for fetching next instruction before Executing the present instruction. ARM has cpi (clock per instruction) of One cycle

6. pipelining: Instructions are broken down and decoded in One pipeline stage. The pipeline advances One step at a time to increase the throughput (Rate of processing).

7. Large number of registers: Large number of registers are used in ARM processor to prevent large amount of Memory interactions. These act as local Memory store for all operations.

→ ARM processor families:

Architecture version	processor families	Processor	Features
ARMv 4T	ARM7TDMI	ARM720T	Von-Neumann 3-stage pipeline
	ARM9TDMI	ARM920T ARM922T ARM 942T	MMU, Harvard 5-stage pipeline
ARM v5 TE, ARM v5 TEJ	ARM 9E	ARM926EJ-S,	MMU, DSP, Jazelle
		ARM946E-S)	MPU, DSP
		ARM 966 HS	MPU (optional), DSP
	ARM10E	ARM1020 E	MMU, DSP
		ARM1026 EJ-S	MMU/MPU, DSP, Jazel

ARMV6	ARM11	ARM1136 T2(F)-S	MPU, DSP
		ARM1176 JZ(F)-S	MMU, Trust Zone, DSP, Jazelle
		ARM11 MP core	MMU, Multiprocessor Cache Support, DSP, Jazelle
ARMV6-M	Cortex	Cortex-M0	NVIC
		Cortex-M1	FPGA TCM interface, NVIC
ARMV7-M	Cortex	Cortex-M3	MPU (optional), NVIC
ARMV7-R	Cortex	Cortex-R4	MPU, DSP
		Cortex-R4F	MPU, DSP, floating point
ARMV7-A	Cortex	Cortex-A8	MMU, Trust zone, DSP, Jazelle, Neon, floating point
		Cortex-A9	MMU, Trust zone, Multi-processor, DSP, Jazelle, Neon, floating point

ARM XYZTDMIEFJFS

X – Series

Y – MMU

Z – Cache

T – Thumb

D – Debugger

M – Multiplier

I – Embedded ICE – circuit Emulator (ICE) macrocell

E – Enhanced instructions for DSP

J – Java Acceleration by Jazelle

F – floating point

S – Synthesizable version

→ ~~ARM~~ CORTEX - M Series family: [Cortex-M → Cortex micro

1. CORTEX-M Series are used in motion sense devices, Smart
Smart lighting, Automative, Retail & Healthcare industries, Energy g

2. CORTEX-M processor family is based on the M- Architecture
Provides Low-latency & highly deterministic operation for Embedd
Systems.

a) CORTEX-M55:

ARM CORTEX-M55 processor is ARM's most AI Capa
CORTEX-M processor & it Uses ARM Helium Vector processing
technology"

b) CORTEX-M35P:

A Tamper-resistant CORTEX-M processor with optiona
Software isolation Using Trust Zone for ARMV8-M

c) CORTEX-M33:

ARM CORTEX-M33 processor has real-time determin
efficiency & Security.

d) CORTEX-M23:

ARM CORTEX-M23 is the Smallest & Low power micro
troller with trust zone Security.

e) CORTEX-M7:

It is the Energy efficient, highest performance proc

f) CORTEX-M4:

ARM CORTEX-M4 processor implements a good blend of Contro and performance for mixed-signal devices

g) CORTEX-M3:

ARM CORTEX-M3 processor is the industry leading "32-bit Processor" for highly deterministic real-time applications.

h) CORTEX-M1:

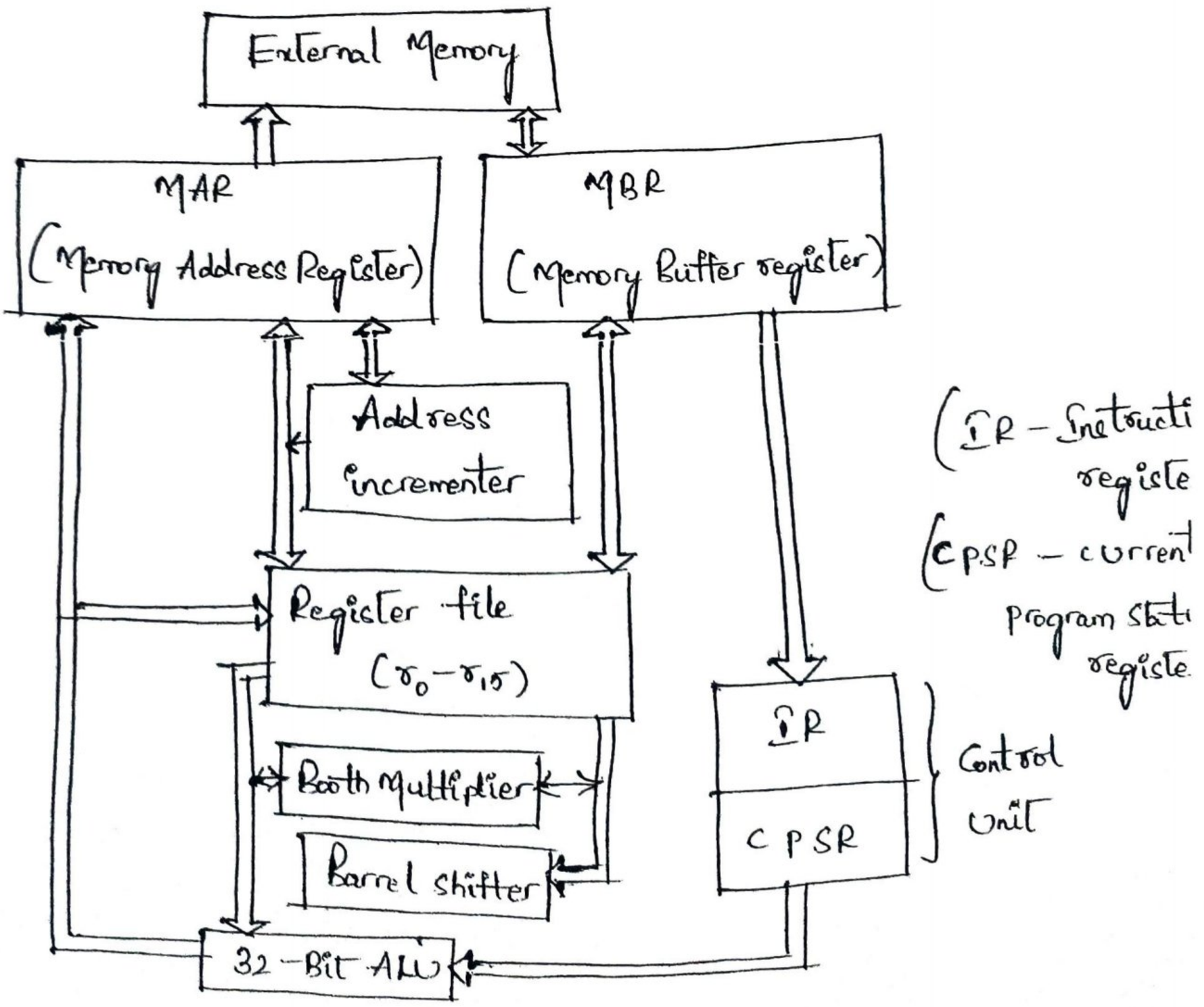
ARM CORTEX-M1 processor targets "FPGA" devices.

i) CORTEX-M0+:

It is the most Energy Efficient ARM processor.

j) CORTEX-M0: It is the Smallest ARM processor.

→ ~~ARM~~ Architecture:



ARM Architecture Consists of

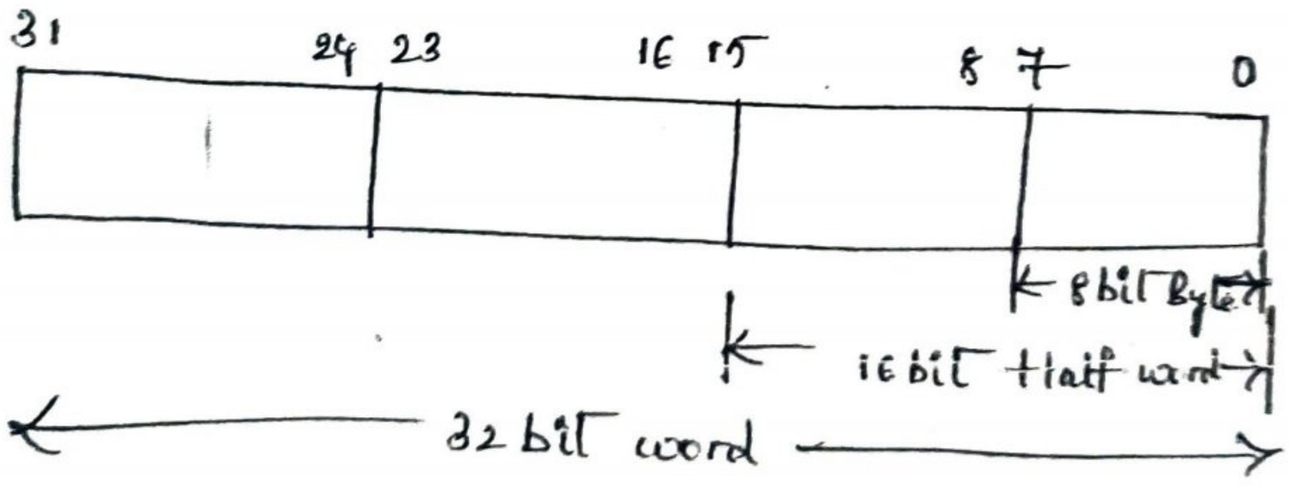
1. Load / Store Architecture
2. A large Array of Uniform Registers
3. Fixed length 32-bit instructions
4. 3-Address instructions

a) Registers:

Only 16 registers are visible to Specific mode. A mode could access

1. A particular set of $r_0 - r_{12}$
2. r_{13} (Stack pointer, SP)
3. r_{14} (Link Register, LR)
4. r_{15} (Program Counter, PC)
5. Current program Status Register (CPSR)

General-purpose Registers:

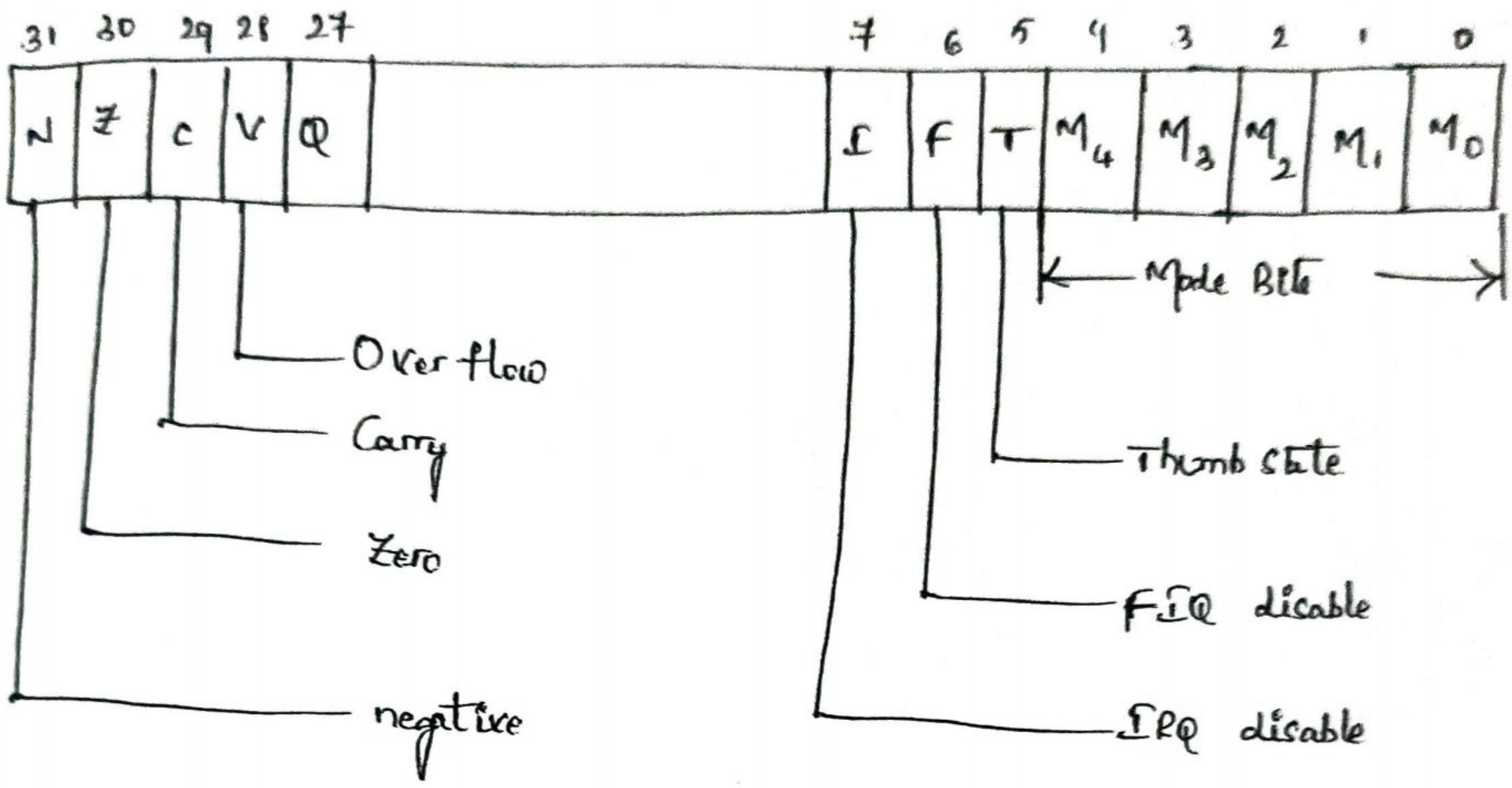


1. 6 data types (Signed / Unsigned)
2. All ARM operations are 32-bit. Shorter data types are only Supported by data transfer operations

b) Program Counter:

1. Store the address of the instruction to be executed
2. All instructions are 32-bit wide and word Aligned.
3. Last two bits of PC are Undefined.

c) program Status Register (PSR):



c) Processor Modes:

Processor Mode	Description
User (usr)	Normal program Execution mode
FIQ (fiq)	Supports a high Speed data transfer
IRQ (irq)	Used for general purpose interrupt handling
Supervisor (svc)	A protected mode for the operating system
Abort (Abt)	Implements virtual memory
Undefined (Und)	Supports Software Emulation
System (sys)	Runs previliged System tasks.

Barrel Shifter:

A Barrel shifter is a digital circuit that can shift a data word by a Specified number of bits without the use of Sequential logic

Booth Multiplier:

Booth Multiplier uses Booth Multiplication Algorithm It is used for both Multiplication and division operations used for Load and Store instructions.

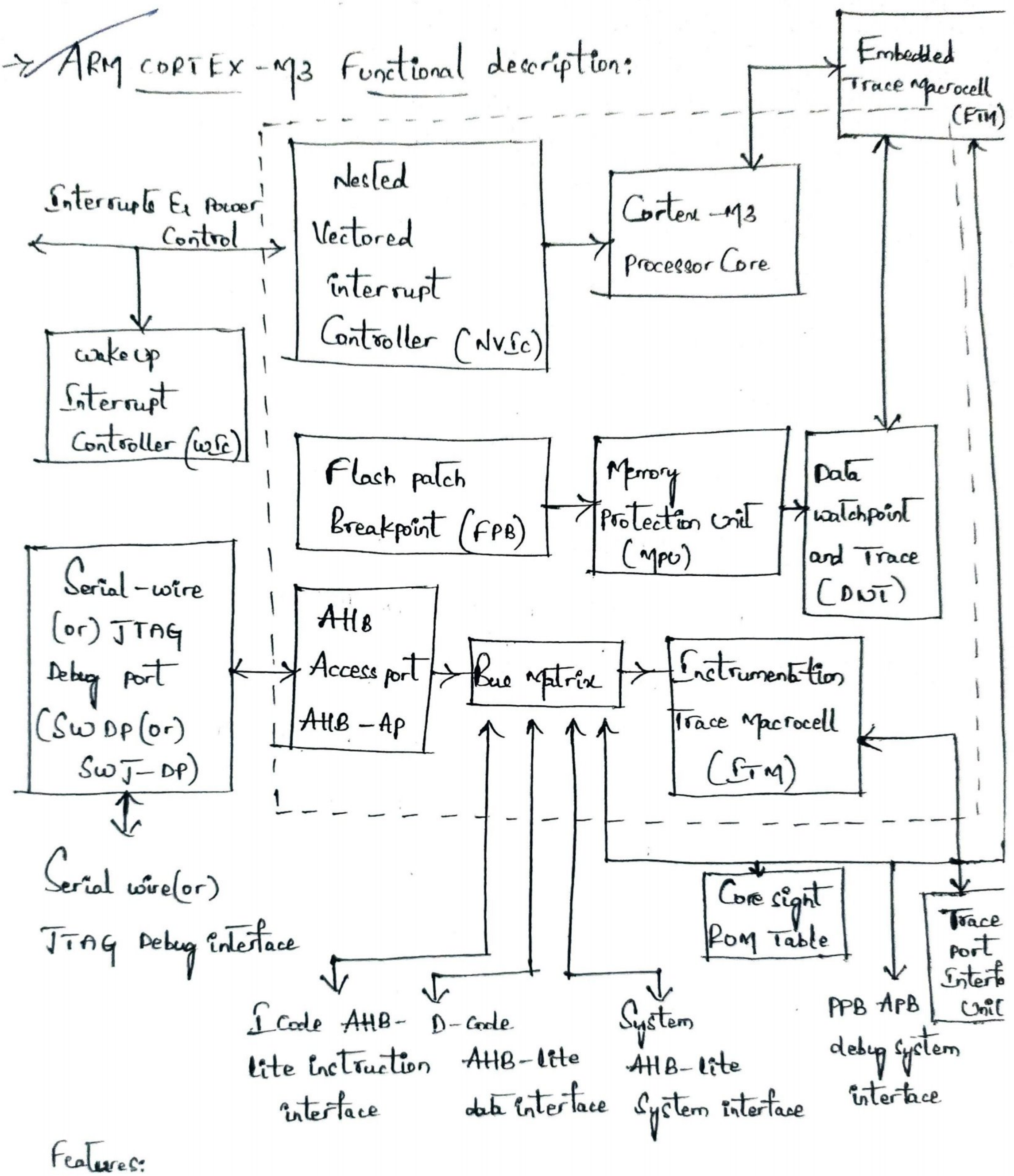
Memory Address Register (MAR):

It holds the address of the Memory location to (or) from which the data is to be transferred.

Memory Buffer Register (MBR):

It is Used to store addresses which acts as Connection between Memory address & Memory data Register.

ARM CORTEX-M3 Functional description:



Features:

1. A low Cost processor Core, with low latency interrupt processing that has:
   a) Banked stack pointer (sp)

b) Hardware divide instructions, SDIV & UDIV

c) Thumb and Debug states

d) Support for interruptible Continued instructions LDM, STM, PUSH & POP for low interrupt latency

e) Automatic processor state Saving & restoration for low latency interrupt Service Routine (ISR).

f) Support for ARMV6 big Endian byte invariant (or) little Endian process.

2. Nested Vectored interrupt Controller: (NVIC).

NVIC closely integrated with the processor Core to achi low latency interrupt processing. Features include.

a) External interrupts, Configurable from 1 to 240.

b) Bits of priority, Configurable from 3 to 8.

c) Dynamic reprioritization of interrupts

d) Support for tail-chaining and late arrival of interru This Enables back-to-back interrupt processing witho overhead of state.

e) optional wake-up interrupt Controller (WIC), providi Ultra-low power sleep mode.

3. Memory protection Unit (MPU): It Consists of

a, Eight Memory regions.

b, Sub Region Disable (SRD), Enabling efficient Use o

Memory regions.

4. Bus interfaces: a) Three advanced High-performance Bus lite (AHB-Lite) interfaces: I Code, D Code & System bus interfaces.

b) private peripheral Bus (PPB) based on Advanced peripheral Bus (APB) interface.

c) Bit-band Support that includes atomic bit band write & Read operatio

d) Memory Access alignment

e) Exclusive access transfers for Multiprocessor Systems.

5. Low Cost Debug solution features:

a) Debug access to all Memory and registers in the system, including access to Memory mapped devices, when the Core is halted access to internal Core registers and access to debug Control registers ever "SYSRESETn" is asserted.

b) Serial wire - Debug port (SW-DP) or Serial wire JTAG Debug Port (SWJ-DP) debug access.

c) "optional flash patch & Breakpoint (FPB)" Unit for implementing Breakpoints & Code patches.

d) "optional Instrumentation Trace Macrocell" (ITM) for Support of "printf" style debugging

e) "optional trace port interface Unit" (TPIU) for bridging to a trace Port Analyzer (TPA), including Single wire output (SWO) mode.

f) "optional Embedded Trace Macrocell" for instruction trace.

→ Modes of operation and Execution of ARM:

ARM processor Supports "THREAD" and "HANDLER" operating Modes and may be run in "THUMB" (or) "DEBUG" operating states. In addition, the processor Can limit (or) exclude access to Some resources by executing Code in "previliged" (or) "Unpreviliged" Mode.

1. Operating Modes:

The Conditions which Cause the processor to enter into Thread (or) Handler mode are as follows:

a) The processor enters "thread Mode" on Reset, (or) as a result of an "exception" return. "previliged and Unpreviliged Co Can run in "Thread" mode.

b) The processor enters "Handler mode" as a result of excep All Code is "previliged" in handler mode.

2. Operating states:

The processor Can operate in thumb (or) Debug state:

a) Thumb state: This is normal Execution while running on 16 bit & 32-bit aligned thumb instructions.

b) Debug state: this is the state when the processor is in "Debug".

→ NVIC Programmers Model:

Address of NVIC registers:

0x E000E004	
0x E000E100	— 0x E000E11C
0x E000E180	— 0x E000E19C
0x E000 E200	— 0x E000 E21C
0x E000 E280	— 0x E000 E29C
0x E000E300	— 0x E000E31C
0x E000E400	— 0x E000E41F

→ Loops, Stack and Stack pointer:

Loops:

1. A loop is a block of statements that are repeatedly executed until a condition is satisfied.
2. Assembly language uses "JMP" instruction to implement loops
3. "Loop" instruction extracts the number of iterations from the "ECX" register.
4. Loop instruction decrements the value of "ECX" and compares it with zero.
5. If the value in the "ECX" is equal to zero, the program jumps to the loop; otherwise the program exits the loop.

Eg:

```
MOV   X5, #1
MOV   X4, #100
```

[start index = 1]

[End index = 100]

↓

for loop : // ... Code

// .... Code

↓

ADD X5, X5, #1 [increment index by 1]

↓

CMP X5, X4 [check whether End index has

X5 <= 100

↓

B.LE for_loop

↓ X5 > 100

// . . . Code

Stack:

1. The processor Uses "full descending stack" the stack pointer holds the address of the last stacked item in Memory
2. When the processor pushes a new item On to stack, it decrements the stack pointer & writes the item to the new Memory location.
3. The processor implements two stacks, the "Main stack" and the "process stack", with a pointer held in independent registers.
4. In "THREAD" mode, the Control register Controls whether the processor uses the "Main stack" (or) the "process stack", in "Handler" mode, the processor Uses "Main stack."

Processor Mode	Used to Execute
THREAD	Applications
HANDLER	Exception handlers

Stack pointer:

1. A "stack pointer" is a small register that stores the address of the last program request in "stack"
2. "Stack" is a Specialized "buffer" which stores data from top down
3. A stack also called "PUSH DOWN STACK" operates in last-in-first-out.
4. When a new data enter (or) pushed onto the top of stack, the Stack pointer "increments" to the next Memory address.
5. When a data is pulled (or) popped from the top of stack, the data is copied from the address of stack pointer, and the stack pointer "decrements" to the next available item at the top of stack.

Subroutine and parameter passing:

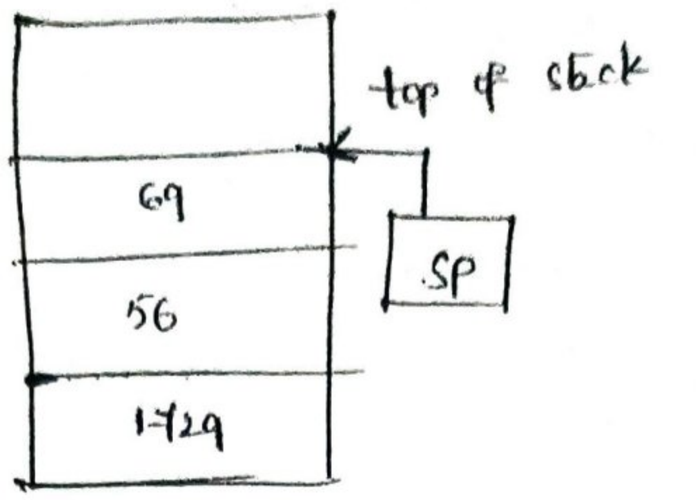
Subroutine:

Group of instructions which are written Separately to perform a Specific task is called "Subroutine"

1. passing parameter via stack:

To pass parameters to a Subroutine, the calling program

Pushes them on the stack in the reverse Order so that the last parameter to pass is the first one pushed and the first parameter to pass is the last One pushed



2. passing parameters via Registers:

1. when the Subroutine called, it simply access the right register to pass the parameter.

2. By putting the address of Some parameter in memory in an address register, the Subroutine neednot access memory in Order to fetch the parameters.

3. passing parameters via Memory locations:

1. It is normally used to pass data to Memory mapped i/o devices, as the interrupt handlers for these devices expect the data transfer to be in a given location, which prohibits the use of stack.

Eg: Mov d0, MAILBox (put value of D_0 in Memory)

bsr postMAN

...

POSTMAN : Mov MAILBox (fetch the data from Memory locatio

...

rts

→ Programming techniques:

ARM CORTEX - M3 can be programmed using either "Assembler" or "C" there might be Compilers for other languages but mostly used Assembler, C, (or) a Combination of both.

1. Using Assembly:

for Small projects, it is possible to develop the whole application in Assembly language. In addition, handling Complex data structures (or) function library management is difficult in Assembler.

1. Functions that Cannot be implemented in 'C', Such as Special register accessed.
2. Timing Critical routines.
3. Tight Memory requirements.

2. Using C:

1. 'C' has the advantage of being portable & Easier for implementing Complex operations.
2. It is a Generic Computer language, 'C' does not Specify how the processor is initialized.

Eg: Using Assembly:

```
START
    MOV   r0, #10
    MOV   r1, #3
    ADD   r0, r0, r1
```

} Application Execution

```
MOV   r0, # 0x18
MOV   r1, = 0x20026
SVC   # 0x123456
END
```

} Application termination

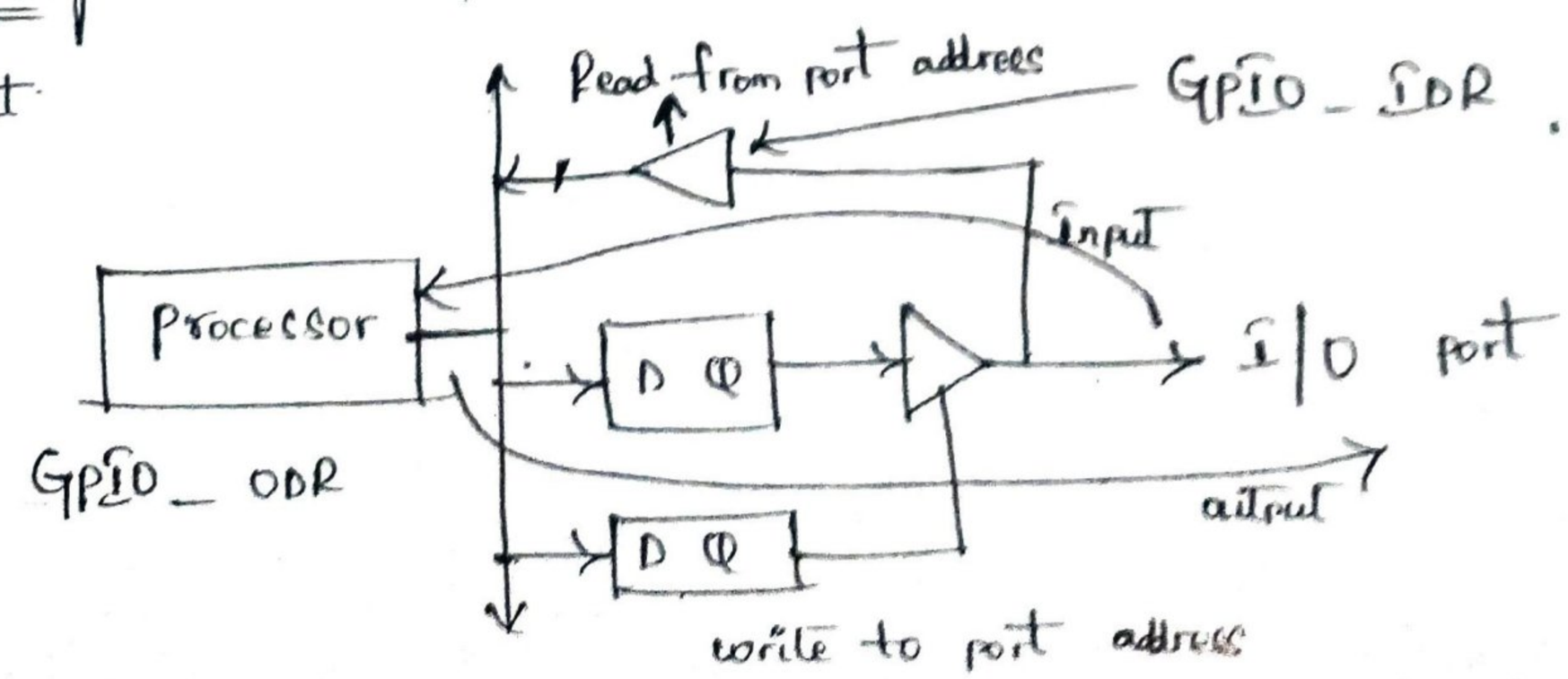
——— program End

Eg: Using C

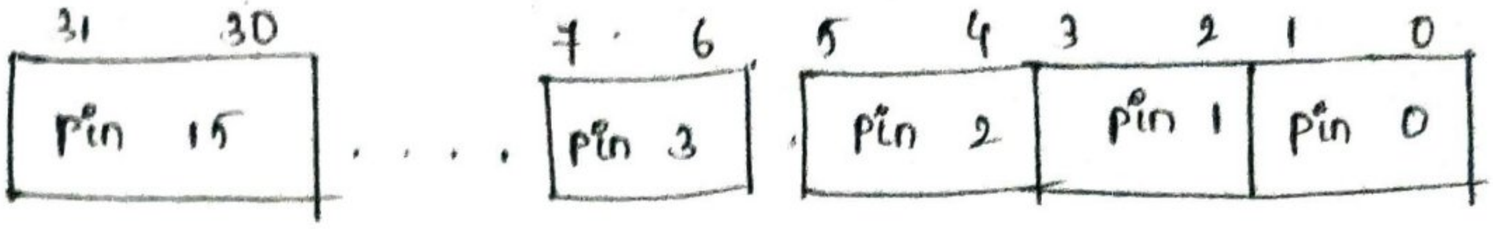
```
i = 10
PRINT " * ";
i = i+1
```

→ Parallel I/O:

1. "parallel port," multiple bits read | written in parallel by CPU
2. "parallel input port", CPU can access information from an external device
3. "parallel output port," CPU can send information to an external device
4. Multiple I/O ports in most ARM CORTEX-M support additional I/O ports via expansion buses.
5. Each port is configured and accessed via one (or) more registers
6. Each register is assigned Unique Memory address
7. CPU reads/ writes data via "data registers" in the port hardware
8. "Mode Register" for a port determines whether each pin is input (or) output



Mode Register:



1. (2) bits per pin.

00 — input mode (Reset state)

01 — General purpose output mode

(IDR - input data register)

(ODR - output data Register)

1. write pin value to ODR
2. Read IDR to determine pin state
3. Read ODR for last written value

10 — Alternate function mode

11 — Analog Mode

→ Disable o/p Buffer, input schmitt trigger, pull resistors.

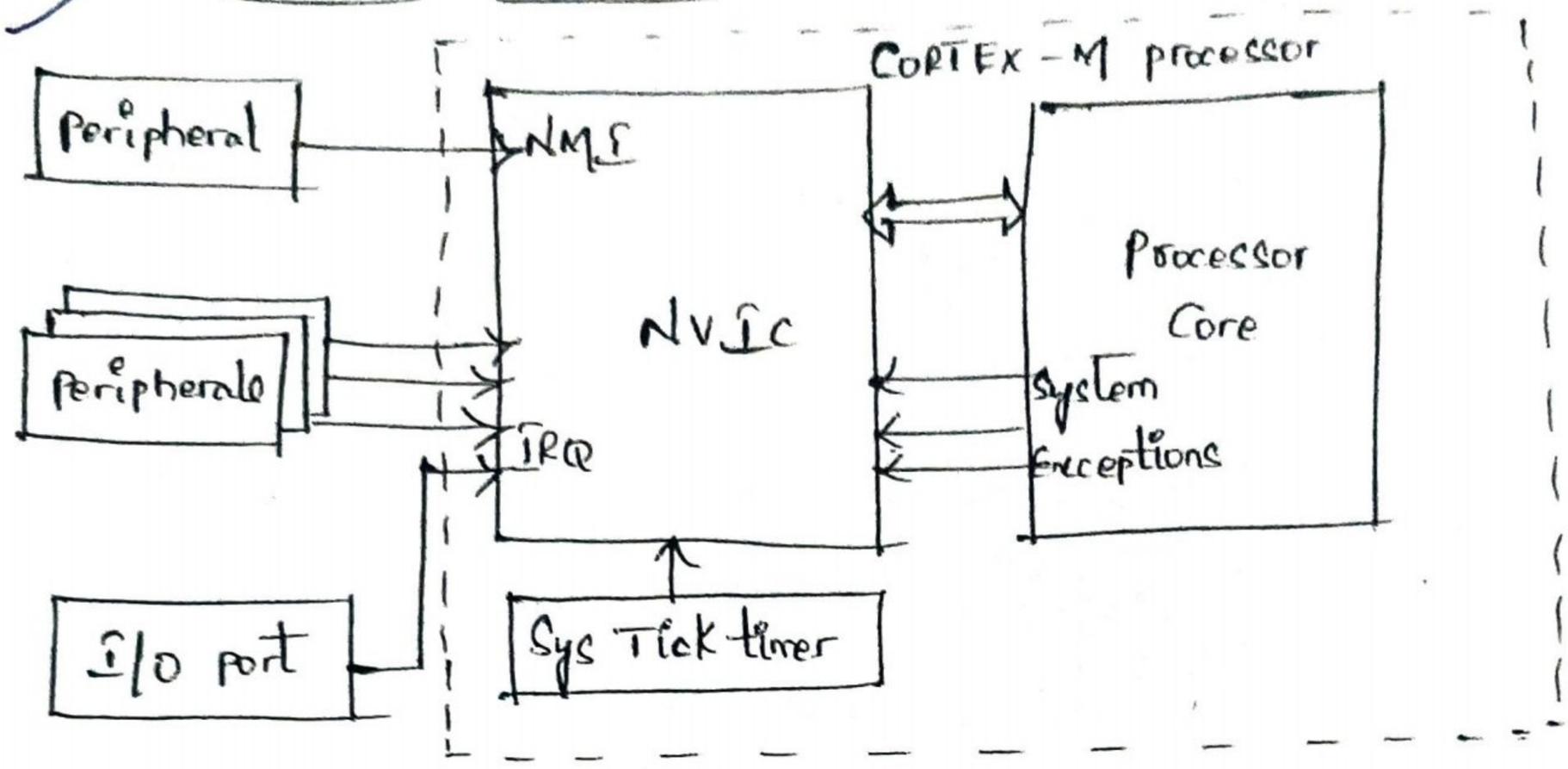
Data Registers: a) IDR:

1. Data input through 16 pins.
2. Read only

b) ODR:

1. write data to be output to 16 pins.
2. Read last value written to ODR.
3. Read / write (for Read - Modify - write operations)

→ NVIC functional description:



1. Nested Vector interrupt Controller (NVIC) is a method of prioritising interrupts, improving the MCU performance and reducing "interrupt latency"

2. NVIC provides implementation for handling interrupts that occur when other interrupts are being executed (or) when the CPU is in process of restoring its previous state & resuming its suspended proces

3. The term "nested" refers to, in NVIC a number of interrupts can be defined, and each interrupt is assigned a priority, with "0" being the highest priority.

4. Function of "NVIC" is higher priority interrupts are completed before lower priority interrupts, even if the lower priority interrupt is triggered first

5. The term "vector" refers to, in which the CPU finds the program, or ISR, to be executed when an interrupt occurs.

6. NVIC uses a vector table that contains addresses of the ISRs Each interrupt.

7 when the interrupt is triggered, the processor gets the address from the Vector table.

8 prioritization & handling Schemes of NVIC reduce low power Consumption, even with high interrupt loading on the Microcontroller.