UNIT-II

Hardware Components - Computing (Arduino, Rasberry-pi):

IOT is no longer a buzzword. With several inspiring use cases, emanating daily, multiple firms are now discovering how they could support the technology for business growth.

It is fast becoming an important featur for new devices to be IOT based, irrespective of the other technologies implemented.

Each is a part of an IOT hardware platform - a combination of hardware, connectivity-tools and software development environment for IOT projects.

Arduino & raspberry pi are not only and the best IOT platforms worth knowing. In fact there are dozens of platforms with a diverse choice of hardware, support, development infrastructure, and communities.

Arduino hardware is an affordable and easy-to set up option for building a basic IOT device that is supposed to perform one action, for example, read humidity sensor data. Arduino community is one of the oldest in this domain, so there won't be lack of support (or) resources. On top of that, Arduino's functionality is easily expandable with on-top sheilds and multiple digital and analog general purpose I/O pins.

Raspberry-pi is the best choice for data-heavy connected devices like hubs, gateways, datum collectors, (or) personal cloud servers; however, it will also be a good fit for simpler IOT applications.

Arduino - Basics

Arduino is a prototype platform (open-Source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programmed (referred to as a microcontroller) and ready-made software called Arduino-IDE (Integrated development environment), which is used to write and upload the computer code to the physical board.

The Key features are:

→ Arduino boards are able to read analog (or) digital input signals from diffrent sensors and turn it into an output such as activating a motor, turning LED ON/off, connect to the cloud and many other actions.

→ control board functions by sending a set of instructions to the microcontroller on the board via Arduino IDE.

→ Arduino does not need an extra piece of hardware in order to load a new code onto the board. we can simply use a USB cable.

→ Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program.

→ Finally, Arduino provides a standard form factor that breaks the functions of the microcontroller into a more accessible package.

Various kinds of Arduino boards are available depending on diffrent microcontrollers used. However all arduino boards have one thing in common: they are programmed through the Arduino IDE.

Some boards are designed to be embedded and have no programming interface (hardware) which need to buy separately.

Arduino boards based on ATMEGA328 Microcontroller

Board name	operating voltage	clock speed	Digital I/O	Analog I/O	PWM	UART	programming interface.
Arduino UNO - R3	5V	16MHz	14	6	6	1	USB via ATMEGA16U2
Arduino-UNO R3 SMD	5V	16MHz	14	6	6	1	USB via Atmega 16U2.
Red Board	5V	16MHz	14	6	6	1	USB via FTDI
Arduino-pro 3.3V | 8MHz	3.3V	8MHz	14	6	6	1	FTDI-compatible Header.
Arduino-pro 5V | 16MHz	5V	16MHz	14	6	6	1	FTDI compatible Header
Arduino-mini-05	5V	16MHz	14	8	6	1	FTDI compatible header
Arduino-pro mini 3.3V | 8MHz.	3.3V	8MHz	14	8	6	1	FTDI compatible header.
Arduino-pro mini 5V | 16MHz	5V	16MHz	14	8	6	1	FTDI compatible header.
Arduino-Ethernet	5V ~~3.3V~~	16MHz	14	6	6	1	FTDI compatible header.
Arduino-FIO	3.3V	8MHz	14	8	6	1	FTDI compatible header.
lilly pad Arduino 328-main board	3.3V	8MHz	14	6	6	1	FTDI compatible header.

Raspberry-pi

The Raspberry-pi is a single board computer developed by Raspberry-pi foundation. It is widely popular as a small, inexpensive computing board among experimenters, hobbyists, educators, and technology enthusiasts.

While the Raspberry-pi is naturally a general purpose device, it will be an injustice to ignore the contribution of the raspberry to the developement of some of the IOT products and projects currently in popular.

They are generally too robust and sophisticated to be used in the development of simple connected sensors (or) actuations, but they find applications serving as data aggregators, hubs, and device gateways in IOT projects.

The latest of the raspberry pi boards, the raspberry-pi 3 model B+ features a 1.4 GHz Broadcom BCM2837B0, Cortex A53 (ARM V8), 64 bit SOC, 2.4 GHz and 5GHz IEEE 802.11. b/g/n/ac wireless lan, Bluetooth 4.2, BLE, and a Gigabit Ethernet port over USB 2.0 (maximum throughput is

300 Mbps). Asides from several other fealeeres including 4 USB ports, Audio output, the board comes with a 1 GB LPDDR2 SDRAM which makes it quite fast for IOT based tasks.

① Embedded processors:

processors are used in majority of electronic products. eg: Mobile phones, televisions, washing machines, cars, smart cards have processors inside.

In most cases, these processors are placed inside in chips called 'Microcontrollers'.

In modern microcontrollers, the chip also contains the essential elements like memory systems, and interface hardware (peripherals).

There are many different types of microcontrollers they can be available with different processors, memory sizes, peripherals inside can be available in different packages.

Large number of microcontrollers are designed for general purpose, which means they can be used in wide range of applications.

Some times, processors are used in chips that are designed for specialized purposes and for particular products, and they are referred as 'Application Specific Integrated Circuits. (ASICS).

In some chip product designs, the chip could be referred to as 'system-on-chip (SOC). This SOC can ranged from very complex application

processor designs for mobile computing to use low-power designs.
[Ex: a smart phone's application processor chip can contain a number of processors.]

Recall processor, cpu, core, Microprocessor & Micro controllers :

→ The term CPU (central processing unit) is referred to the main processor chip used in a computer, usually inform of a physical chip product, which requires external memory chips.

The Term CPU is used frequently to day, but the word 'central' might no longer be relevant the word to a number of systems because may systems contain multiple processors. we normally just refer the processing unit as a 'processor'.

processor core/ CPU core :

Typically refers to the processor inside a micro controller product (or) chip product, excluding the memory system, peripherals, and other system components.

The word 'core' might also refer to the part inside a processor that handels software execution, excluding the interrupt controller and debug software.

Micro processor: a chip device containing processor, which is designed primarily to handle computational tasks, and can also handle control tasks. The System designers typically need to add memory and potentially additional peripheral hardware to build a complete system with microprocessors.

Micro controller: a chip device containing processors' which is designed to handle control and computational tasks. This chip typically contains a memory system, (flash memory for program ROM, SRAM) and a number of peripherals.

understanding Different types of processors.

As a result, we need to have different types of processors for diffent applications. Based on, the technical requirements of the applications, chip designers need to select the right processor for the project, and sometimes need to compromise between various requirements to create designs that fit the targetted applications.

Fortunately there are many diffent types of processors available on the market, in addition to different ~~types of processors avai~~ performance points and sizes, some of them also have special feature to fit certain markets.

For example, ARM provides a wide range of processors that are designed to suite most of the target applications very well by providing the right balance between performances, features and power.

ARM architectural profiles:

Reduced instruction set computing (RISC) has a special place of the map of hardware development and the family of ARM processors is based on RISC Architecture.

ARM has been designing processors ~~fr~~ over 20 years. Most of the processors designed by ARM-32-bit and in last few years ARM also have been developed processors that support a 32-bit & 64-bit architecture.

ARM7TDMI processor (ARM7V) is the first key ARM processor that widely deployed in the market.

It is very energy efficient, and provides high code density using an innovative operation state that support 16-bit instruction set called "Thumb".

As a result, it was used in a number of second generation mobile phones, and a number of microcontroller products.

In around 2003, ARM realized that it needs to diversify the processor products to address different technical requirements in different markets.

As a result, three product profiles are defined, and the cortex processor brand name is created for the naming of these processors:

i. Cortex-A-processors:

These are Application processors, which are designed to provide high performance and include features to support advanced operation systems (eg: Android, Linux, windows, iOS).

These processors typically have longer processor pipeline, and can run at relatively high clock frequency (over 1 GHz).

These processors have Memory Management Unit (MMU) to support virtual memory addressing required by advanced OS, enhanced Java script, and secure program execution environment

The Cortex-A processors are typically used in Mobile phone, Mobile Computing devices (eg: tablets), Television.

(ii) Cortex-R processors:

There are Real-Time, high performance processors that are very good at data crunching, can run at high clock speed (500 MHz to 1 GHz) and at the same time can be very responsive to hardware events.

They have cache memories as well as Tightely-coupled Memories, which enable deterministic behavior for interrupt handling.

The Cortex-R processors are also designed with additional features to enable much higher system reliability such as Error Correction Code (ECC) support for memory systems and dual-core lock-~~st~~step feature.

The Cortex-R processor's can be found in hard disk drive controllers, wireless base-band controllers, specialized microcontrollers such as Automative and Industrial controllers.

Cortex-M processors:

The cortex-M processors are designed for main stream microcontroller market where the processing requirement is less critical, but need to be very low power.

Most of the Cortex-M processors are designed with a fairly short pipeline, for example: two stage in the Cortex-M0+ processor and three stages in cortex-M0, cortex-M3, and cortex-M4. The cortex-M7 processor has a longer pipeline (six-stages) due to higher performance requirement, but still the pipeline is a lot shorter than the designs of high-end application processors.

These Cortex-M processors are slower than cortex-R & cortex-A, due to 100MHz clock speed.

Due to their low power, fairly high performance and ease of use benefits, the cortex-M processors are selected by most microcontroller products.

These processors are used in some of the sensors, wireless communication chip sets, mixed signal ASIC's & SOC products.

Cortex processors and Architecture versions:

V7-A (Applications)	V7-R (Real-time)	V6-M/V-7-M controller
Cortex-A5 (single/MP)	Cortex-R4	Cortex-M0+ (ARMV6-M)
Cortex-A7 (MP)	Cortex-R5	Cortex-M0 (ARMV6-M)
Cortex-A8 (single)	Cortex-R7	Cortex-M1 (ARMV6-M)
Cortex-A9 (single/MP)		Cortex-M3 (ARMV7-M)
Cortex-A12 (MP)		Cortex-M4 ARMV7-M
Cortex-A15 (MP)		

Thumb: The ARM7 architecture contains two instruction sets, the ARM and Thumb instruction sets. The thumb instruction set is a subset of the most commonly used 32-bit ARM instructions. Thumb instructions are each 16-bits long, and have a corresponding 32-bit ARM instruction that has the same effect. Thumb code is to reduce cache density. Because of its improved density, Thumb code tends to cache better than the equivalent ARM code and can reduce the amount of memory required.

Thumb 2: Thumb 2 technology was introduced in ARM V6T2, and is required in ARMV7. This technology extends the original 16-bit Thumb instruction set to include 32-bit instructions.

VFP: The VFP extension was called the Vector floating-point architecture, and supported vector operations. VFP is an extension that implements single-precision and optionally, double-precision-floating-point arithmatic, compliant with the ANSI/IEEE standard for floating-point-Arithmatic

Advanced SIMD (NEON)

The ARM NEON technology provides an implementation of the SIMD instruction set, with separate register files. Some implementations have a separate NEON pipeline back-end. It supports 8, 16, 32 & 64-bit integer, and single-precision (32-bit) floating-point data, that can be operated on as vectors in 64-bit & 128-bit registers.

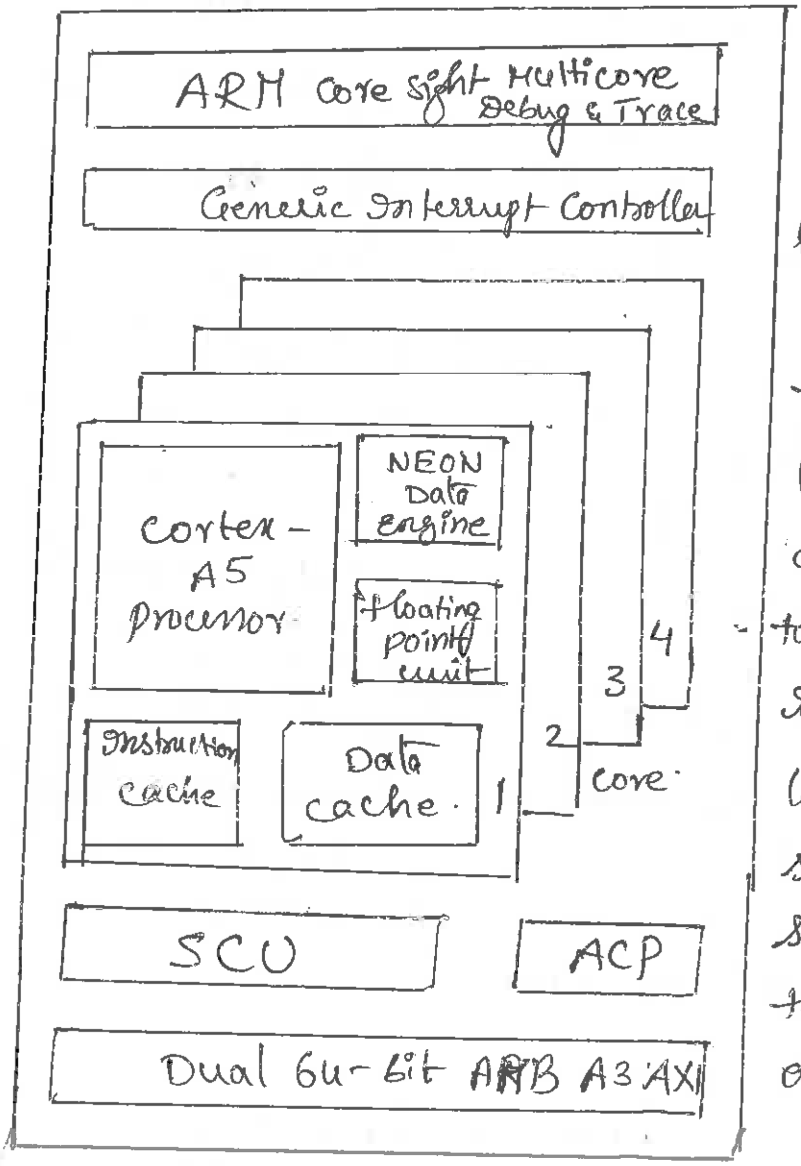
Large physical address Extension:

LPAE is optional in the V7-A architecture and is presently supported by the cortex-A7, Cortex-A12, and Cortex-A15 processors. It enables 32-bit processors that are normally limited to addressing a maximum of 4GB of address space to access upto 1TB of address space by translating 32-bit virtual memory addresses to 40-bit physical memory addresses.

Cortex - A - Series processor properties.

	Cortex A5	Cortex A7	Cortex A8	Cortex A9	Cortex A12	Cortex. A15
Release date	Dec 2009	oct 2011	July 2006	March 2008	JUN 2013	APR 2011.
Typical clock speed	≅ 1GHz	≅ 1GHz on 28nm	≅ 1 GHz on 65nm	≅ 2GHz on 40nm	≅ 2GHz on 28nm	≅ 2.5 GHz on 28nm.
Execution order	In-order	In-order	In-order	Out of order	Out of order	Out of order.
Cores	1 to 4	1 to 4	1	1 to 4	1 to 4	1 to 4.
peak-Integer-Throughput	1.6 DMIPS	1.9 DMIPS/MHz	2 DMIPS/MHz	2.5 DMIPS/MHz	3.0 DMIPS/MHz	3.5 DMIPS/MHz
VFP architecture	VFP V4	VFP V4	VFP V3	VFP V3	VFP V4	VFP$_{V4}$
NEON architecture	NEON	NEON V2	NEON	NEON	NEON V2	NEON V2
Hardware divide.	NO	YES	NO	NO	YES	YES.
Fused multiply Accumulate	Yes	Yes	NO	NO	Yes	yes.
pipe line stages	8	8	13	9 to 12	11	15+
Instructions decoded per cycle	1	Partial dual Issue.	2 superscalar	2 superscalar	2 superscalar	3 superscalar
LAPE	NO	Yes	NO	NO	Yes	Yes
Floating point unit	optional	yes	yes	optional	yes	optional

1. The Cortex-A5 processor:

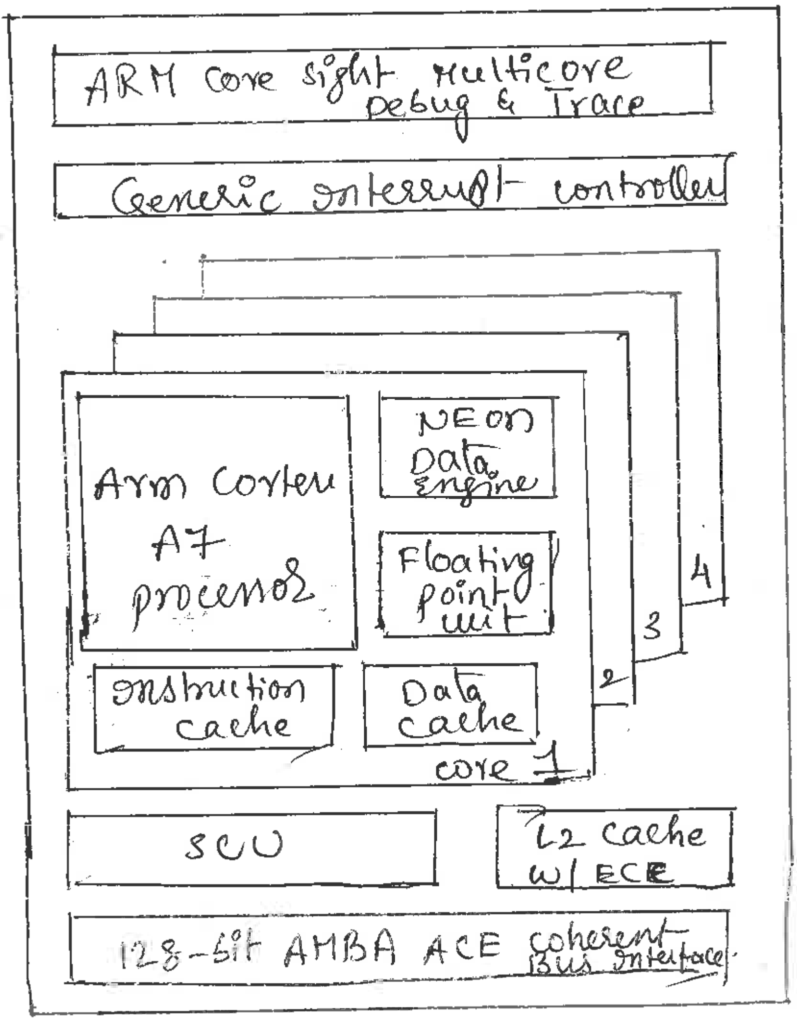


The cortex-A5 processor is the smallest ARM multicore application processor.

Devices based on this processor are typically low cost, capable of delivering the internet to the widest possible range of devices from low-cost entry-level smart phones, and smart mobile devices, to embedded, consumer and industrial devices.

2. The ARM cortex-A7-processor:

The ARM cortex-A7 processor is the most energy efficient application processor developed by ARM and extends ARM's low-power leadership in entry level smart phones, tablet and other advanced mobile devices.



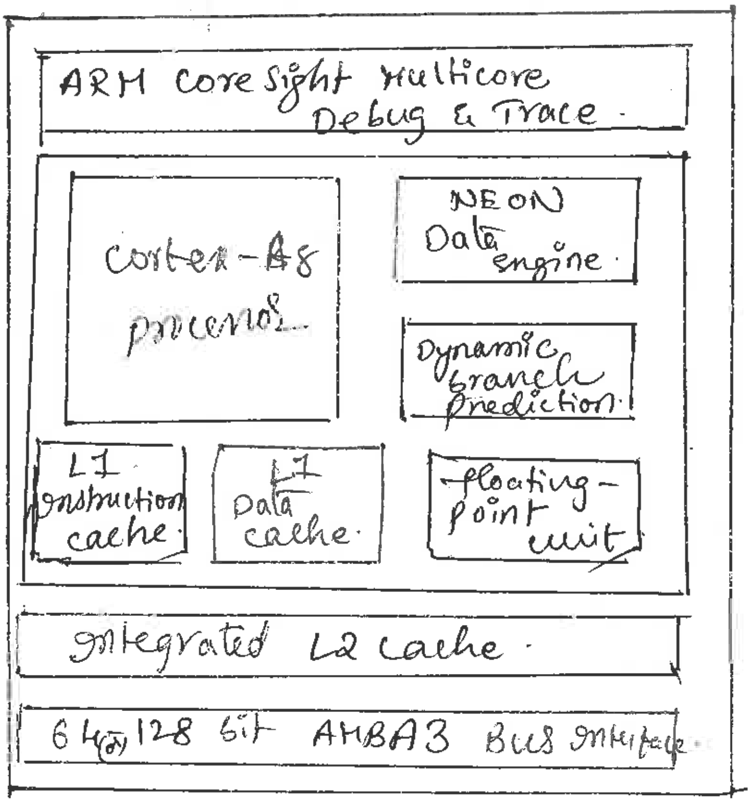
features:

1. Architecture and feature set identical to the cortex-A15 processor.
2. Less than 0.5mm², using 28nm process Technology.
3. Full application compatibility with all cortex-A series processors.
4. Tightly coupled low-latency level-2-cache up to 4 GB.
5. NEON technology for multimedia and SIMD processing.

3. ARM Cortex-A8-processor:

features

1. frequency from 600MHz to more than 1GHz.
2. High performance Superscalar processor.
3. NEON technology for multi-media and SIMD processing.
4. compatibility with older ARM processors.



Key architectual points of ARM Cortex-A Series processors:

1. 32-bit RISC core, with 16 X 32 bit visible registers with mode based register banking.

2. Modified Harvard Architecture (separate, concurrent access to instructions & Data).

3. Load/store Architecture.

4. Thumb-2 technology as standard.

5. VFP and NEON options.

6. Backward compatibility with code from previous ARM processors.

7. 4GB of virtual address space and a minimum of 4GB of physical address space.

8. Hardware Translation table walking for virtual to physical address translation.

9. Virtual pages sizes of 4KB, 64KB, 1MB & 16MB.

10. Big-endian and little endian data access support.

11. unaligned access support for basic load/store instructions.

12. Symmetric Multiprocessing (SMP) support on MP core variants, i.e multicore versions of the Cortex-A series processors, with full data coherency at the L1 cache level.

Registers of ARM processors.

The ARM architecture provides sixteen 32-bit GPR's (R0-R15) for software use. fifteen of them R0-R14 can be used for general purpose data storage, while R15 is the program counter whose value is altered as core executes instructions.

$R_0 - R_7$ → low registers,
$R_8 - R_{15}$ → high registers.
CPSR - current program status register.

processor modes.

	Mode	function.
1.	User	user mode (10000) [unprivileged mode in which most applications run]
2.	FIQ	Fast Interrupt request (10001) [entered on an FIQ interrupt exception]
3.	IRQ	Interrupt request (10010) [entered on an IRQ interrupt exception]
4.	Supervisor (SVC)	Supervisory mode (10011) [entered on reset (or) when a supervisory call instruction is executed].
5	Monitor	(Monitoring mode (10110) [Implemented with security extensions].
6.	Abort	Abort mode [entered on a memory access exception].
7.	Undef	undefined mode [entered when an undefined instruction executed].
8.	SYS	system mode [privileged mode, sharing the register view with user mode].

USR	SYS	FIQ	IRQ	abt	SVC	und	mon
R0	—	—	—	—	—	—	—
R1	—	—	—	—	—	—	—
R2	—	—	—	—	—	—	—
R3	—	—	—	—	—	—	—
R4	—	—	—	—	—	—	—
R5	—	—	—	—	—	—	—
R6	—	—	—	—	—	—	—
R7	—	—	—	—	—	—	—
R8	—	R8 fiq	—	—	—	—	—
R9	—	R9_fiq	—	—	—	—	—
R10	—	R10_fiq	—	—	—	—	—
R11	—	R11-fiq	—	—	—	—	—
R12	—	R12-fiq	—	—	—	—	—
SP-R13	—	SP_fiq	SP_svc	SP_abt	SP_svc	SP_und	SP_mon
LR-R14	—	LR-fiq	LR_svc	$R_abt	$R_svc	LR_und	LR_mon
PC-R15	—	—	—	—	—	—	—
(A/c)PSR	—	—	—	—	—	—	—
		SRSR_fiq	SPSR_IRQ	SPSr_abt	SPSr_svc	SPSr_und	SPSR_mon

a

Instruction set

ARM & Thumb Instructions

In any processor architecture, an instruction includes an opcode that specifies the operation to perform, such as add contents of two registers (or) move data from a register to memory etc., with specified operands, which may specify registers, memory locations (or) immediate data.

ARM architecture has two instruction sets. The ARM instruction set and Thumb instruction set.

In ARM instruction set, all instructions are 32-bit wide and aligned at 4-bytes boundaries in memory.

On the otherhand, in thumb instruction set, all instructions are of 16-bits wide and are aligned at even (or) two bytes boundaries in memory.

(1) Data movement instructions:

MOV r_1, r_2 : move a 32-bit value into a register

MVN r_1, r_3. ; move the NOT of the 32-bit value into a register
move $(\sim r_3)$ to r_1.

Data processing instructions : [Arithmatic]

ADC $r_1; r_2, r_3$; add two 32-bit values and carry.

ADD r_4, r_5, r_3 ; add two 32-bit values.

RSC r_3, r_2, r_1 ; Reverse subtract with carry of two 32-bit values.

$r_3 = r_1 - r_2 \, ! \text{carry}$

RSB r_3, r_2, r_1 ; Reverse subtract of two 32-bit values.

$r_3 = r_1 - r_2$

SBC r_2, r_4, r_6 ; Subtract with carry of two 32-bit values.

$r_2 = r_4 - r_6 - ! \text{carry}$

SUB r_2, r_4, r_6 ; subtract two 32-bit values.

logical instructions

AND r_7, r_5, r_2 ; logical bitwise AND of two 32-bit values

$r_7 = r_5 \wedge r_2$

ORR $r_6, r_4, r_1,$ LSR r_2 ; logical bitwise OR of two 32-bit values

$r_6 = r_4 \vee (r_1 \gg r_2)$

EOR r_5, r_1, r_2 logical exclusive OR of two 32-bit values

$r_5 = r_1 \veebar r_2$

BIC r_3, r_1, r_4 ; logical Bit clear (AND NOT)
$r_3 = r_1 \& \sim r_4$.

comparison instructions.

CMN r_1, r_2 ; compare negated.

CMP r_1, ~~#~~ 0XFF ; compare.

TEQ r_3, r_5 ; Test for equality of two 32-bit values.

TST r_1, r_2 ; Test bits of a 32-bit values.

multiply instructions.

MLA r_1, r_2, r_3, r_4 ; multiply and accumulate
$r_1 = (r_2 * r_3) + r_4$.

MUL r_3, r_7, r_6 ; multiply $r_3 = r_7 * r_6$

Branch instructions:

→ B label ; Branch ; pc = label.

→ BL label ; Branch with link.

→ BX ~~r5~~ ; Branch exchange.

→ BLX r_6 ; Branch exchange with link.

single register transfer :

LDR r0, [r2, #0X8], load register from memory

STR r1, [r4], #0X10 ; store register to memory

multiple register transfer :

LDMIA r6!, {r2 - r4} ; load multiple registers from memory

$r_2 = [r6]$; $r_3 = [r6 + 4]$; $r_4 = [r6+8]$ and update r6 by (r6+12]

STM r1!, {r3-r5} ; store multiple register to memory

$[r_1 - 4] = r_5$

$[r_1 - 8] = r_4$

$[r_1 - 12] = r_3$ and update r_1 by $[r_1 - 12]$

stack operations :

LDMED sp!, {r1, r3} ; $r_1 = [sp + 4]$.

$r_2 = [sp + 8]$

$r_3 = [sp + 12]$

and sp is updated by [sp + 12] .

STMED sp!, {r4, r6} ; store multiple registers to stack memory. $[sp - 4] = r_6$

$[sp - 8] = r_5$

$[sp - 12] = r_4$.

and sp is updated by [sp - 12]

SWAP instruction :

swp/swpB r0, r1,[r2] . ; load a 32-bit word or a byte from the memory address in r2 into r0 and store the data in r1 to the memory address in r2.

program status register instructions.

MRS [MRS r1, cpsR] . move the content of CPSR register to r1 .

MSR [MSR cpsR_f, r1] . update the flag field of CPSR by the content in r1 .

Exception generating instructions :

SWI 0x123456 ; software interrupt for an operating system routine.
change to Supervisor mode,
CPSR is saved in SPSR .
control branches to interrupt vector .

ARM Cortex-M processor Series

Processor	Description
Cortex-M0	The smallest ARM processor - only approximately 12000 logic gates at minimum configuration. It is very low power and energy efficient.
Cortex-M0+	The most energy efficient ARM processor - It is a similar size as the cortex-M0 processor, but with additional system level and debug features, and have high energy efficient than the Cortex-M0 processor design. It supports same instruction set of cortex-M0.
Cortex-M1 :	It is a small processor designed optimised for field programmable Gate Array (FPGA) applications. It has the same instruction set architecture as in the cortex-M0 processor, but has FPGA specific memory system features
Cortex-M3:	when compared to the cortex-M0 & cortex-M0+ processors, the Cortex-M3 has much more powerful instruction set, and its memory system is designed to provide higher processing throughput

	(e.g. use of Harvard architecture). It also has more system level and debug features, but at a cost of larger silicon area (minimum gate count is about 40000 gates) and slightly lower energy efficiency. The cortex-M3 processor is very popular is still a lot better than many traditional 8-bit, 16-bit and have popular 32-bit microcontroller.
Cortex-M4.	The cortex-M4 processor contains all the features of the Cortex M3 processor, but with additional instructions to support DSP applications and have an option to include a ~~FDP~~ Floating point unit (FPU). It has the same system level and debug features as the cortex-M3 processor.
Cortex-M7	It is a high performance processor designed to cover application spaces where the existing Cortex-M3 and cortex-M4 processors can not reach. Its instruction sets is a superset of cortex-M4 processor supporting both single and double precision floating point calculations. It also has many advanced features, which are usually find in high-end processors such as caches and branch predictions.

The applications of various Cortex-M processors.

processor	Application.
Cortex-M0 -M0+	→ General data processing and I/o control tasks. → ultra low power applications. → replacement for 8-bit/16-bit Microcontrollers. → low-cost ASIC, ASSP's.
cortex-M1	→ FPGA applications with small to medium data processing complexity. →
Cortex-M3	→ Feature rich/high-performance/low power microcontrollers. → Light weight DSP applications.
Cortex-M4	→ Feature-rich/high-performance/low-power microcontrollers, DSP applications. ~~Ligh~~ Applications with frequent single-precision floating point operations.
Cortex-M7	→ Feature-rich/very high performance power microcontrollers, → DSP applications. → Applications with frequent single (or) double precision floating operations.

Features	Cortex-M0	Cortex-M0+	Cortex-M1	Cortex-M3	Cortex-M4	Cortex-M7
Number of interrupts	1-32	1-32	1, 8, 16, 32	1-240	1-240	1-240
Interrupt priority levels	4	4	4	8-256	8-256	8-256
FPU	–	–	–	–	single precision	single/double precision.
OS Support	y	y	optional	y	y	y
cache	–	–	–	–	–	optional.
Memory protection unit	–	optional	–	optional	optional	optional.
Debug	optional	optional	optional	optional	optional	yes.
Instruction trace	–	optional MTB	–	optional ETM	optional ETM	optional ETM.

Features of ARM Cortex - M0 :

1. It is 32-bit Reduced Instruction set computing (RISC) processor, based on an architecture specification called ARMV6-M architecture.

2- The bus interface and internal data paths are 32-bit width.

3. Have 16, 32-bit registers in the register bank (r0 to r15), However some of these registers have special purposes.

   R15 - program counter
   R14 - Link register.
   R13 - stack pointer.

4. The instruction set is a subset of thumb-instruction set Architecture. Most of the instructions are 16-bit to provide very high code density.

5. support upto 4-GB address space. The address Space is divided into number of regions.

6. support and designed based on Von-Neumann bus architecture.

7. Designed for low power applications, including architecturmeal support for sleep modes and have various low power features at design level.

8. Includes an Interrupt controller NVIC.
The NVIC provide very flexible and powerful interrupt Management.

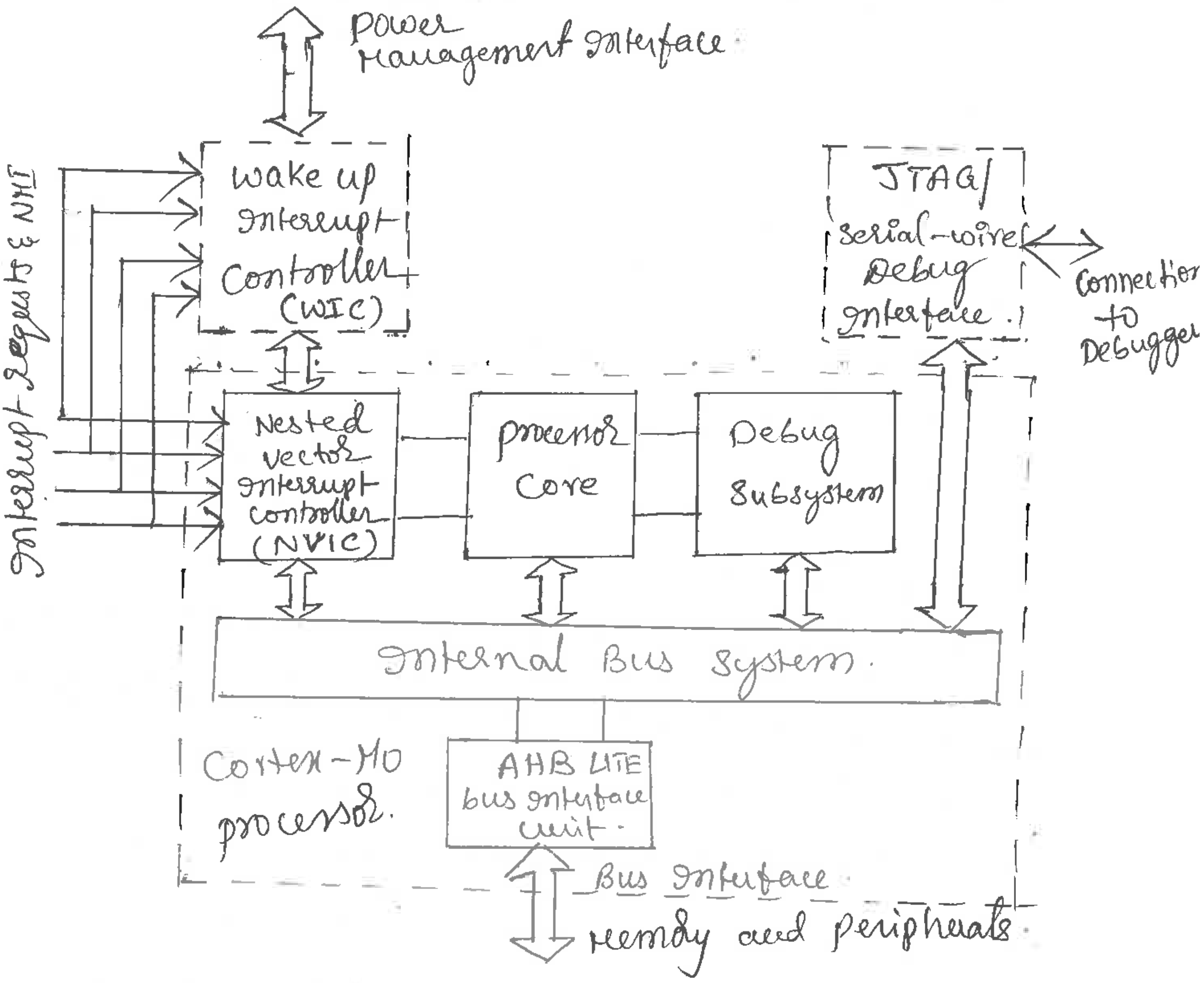
9. The system bus is pipelined, based on a bus protocol called Advanced high performance Bus (AHB) Lite. This bus supports to transfer 8-bit, 16-bit and 32-bit data and also wait states to be inserted.

10. Support various features for OS (operating system) implementation such as a system tick timer, shadowed stack pointer, and dedicated exceptions for OS-operations.

11. Include various debug features to enable software developers to create applications efficiently.

12. provide good performance in most general data processing and I/o control applications.

Block-diagram of Cortex-M0 processor


Power Management Interface
Wake up Interrupt Controller (WIC)
JTAG/ serial-wire Debug Interface
Connection to Debugger
Interrupt Requests & NMI
Nested vector interrupt controller (NVIC)
processor Core
Debug Subsystem
Internal Bus System.
Cortex-M0 processor.
AHB LITE bus interface unit.
Bus Interface
memory and peripherals.

Key charactaristics of Cortex-M0 processor:

1. processor pipe line : The cortex-M0 processor has a three stage pipe line (fetch, decode, execute)

2. Instruction set : The instruction set is ISA Thumb instruction set Architecture. only a subset of thumb ISA is used (56 of them). most of instructions are 16-bit, very few are 32-bit

→ support optional single cycle 32-bit X 32-bit multiply, (or) a smaller multicycle multiplier ~~for~~ designs.

Memory addressing

1. 32-bit addressing supporting upto 4GB of memory space.

2. The system bus interface is based on an on-chip bus protocol called AHB-lite supporting 8-bit, 16-bit and 32-bit data transfer.

Interrupt handling:

The processor includes NVIC, (Nested vector interrupt controller) unit handles interrupt prioritization and masking ~~functions~~.

It supports upto 32-interrupt requests from various peripherals, an additional non-maskable interrupt (NMI input), and also support a number of system exceptions.

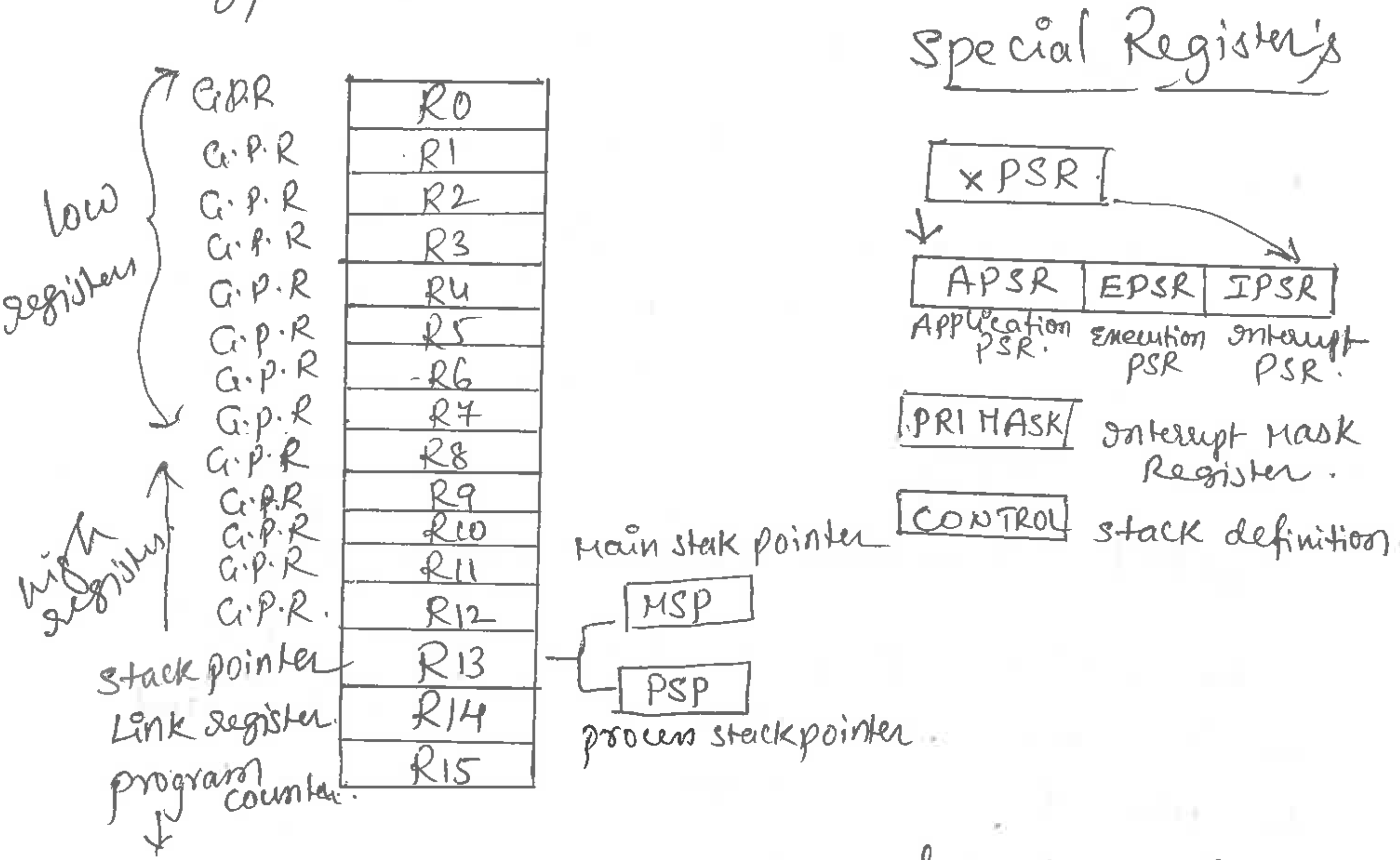
Operating system support.

→ Two System exception types SVcall & PendSV are included to support OS operations.

-An optional 24-bit hardware timer called systick (system-tick-timer) is also included ~~for~~ periodic OS time keeping.

The procend core contains the register banks, ALU, data path, and control logic.

The register bank has sixteen 32-bit registers.



→ Registers R0-R12 are general purpose uses. Due to limited space in the 16-bit Thumb instructions many of the thumb instructions can only access R0-R7, also called low-registers.

→ R13 - stack pointer : It is used for accesing stack memory via push and pop operations. In this processor have two stack pointers are

MSP - main stack pointer is the default stack pointer after Reset, and is used when running exception handlers.

PSP- process stack pointer can onlybe used in Thread mode (not handling exceptions).

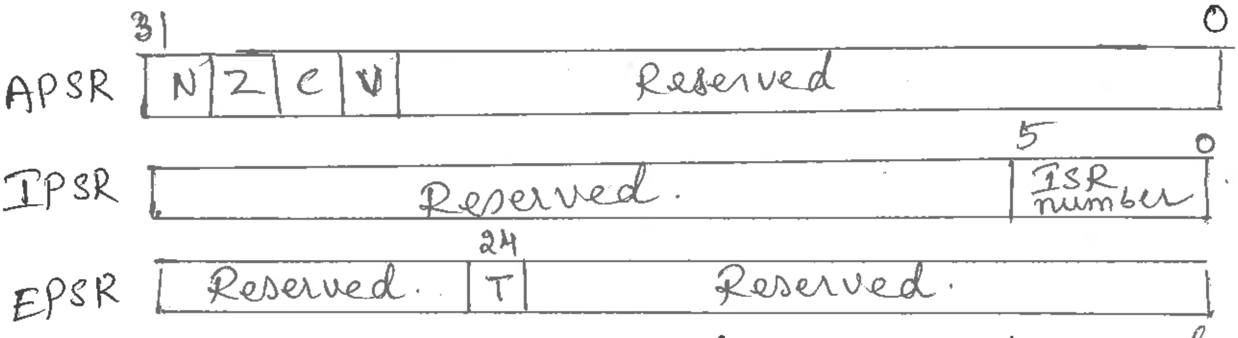
R14- Link register used for storing the return address of a subroutine (or) function call. when BL (or) BLX is executed, the return address is stored in LR.

R15- program counter. It is readable and writable. A read returns the current instruction address plus four. writing to R15 will cause a branch to take place.

xPSR, combined Program status Register.

The combined program status Register (PSR) provides information about program execution and the ALU flags. It consists of the following three PSR's

1. Application PSR (APSR)
2. Interrupt PSR (IPSR)
3. execution PSR (EPSR).



N- negative flag, Z-zero flag, C- carry/borrow flag
V- overflow flag.
T- If T=1, Support Thumb state.

ISR number: current executing ISR number.

PRIMASK :

The PRIMASK register is a 1-bit wide interrupt mask register. when set, it blocks all interrupts apart from NMI and the Hard fault exception.

The PRIMASK register can be accessed using special register access instructions (MSR & MRS) as well as using an instruction called CPS.

→ The NVIC accepts up to 32 interrupt request signals and a NMI input. If an interrupt is accepted, the NVIC communicates with the processor so that the processor can execute the correct interrupt handler.

→ The WIC is an optional unit. In low power applications, the microcontroller can enter stand by state with most of the processor powered down. under this situation, the WIC can perform the function of interrupt masking while NVIC ~~options~~ and the processor core are inactive.

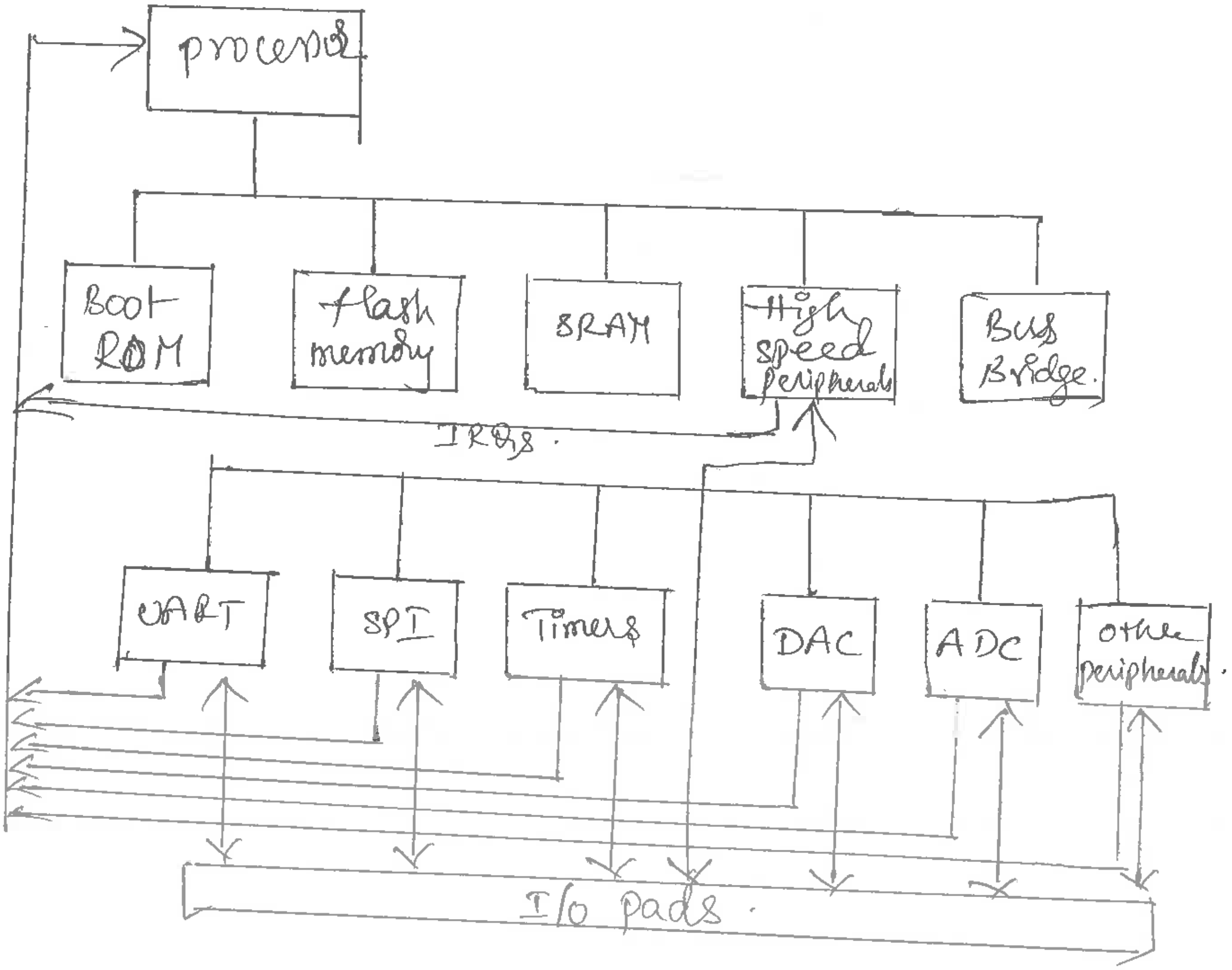
when interrupt request is detected, the WIC informs the power management to power-up the system so that the NVIC & the processor core then handle the rest of the interrupt processing.

→ The debug subsystem contains various functional blocks to handle debug control, program break points, and data watch points. when a debug~~ge~~ event occurss, it can put the processor core in a halted state to that embedded developers can examine the status of the processor at that point.

→ AHB-lite is an on-chip bus protocol used in many ARM processors. This bus protocol is part of the AMBA specification, which is a bus architecture developed by ARM and widely used in IC design industry.

→ The JTAG (or) serial wire interface units provide access to the bus system and debugging functionalities. The JTAG protocol is a popular 4-pin communication protocol commonly used for IC and PCB Testing.

The serial wire protocol is a newer communication protocol that only requires two ~~units~~ wires, but it can handle the same debug functionalities.



A simple System with Cortex-M0 processor.

commonly used directives for inserting data into a program :

1. Byte.	'DCB' eg: DCB 0X12.
2. Halfword	'DCW' eg: DCW 0X1234.
3. word	'DCD' eg: DCD 0X01234567.
4. Double word	DCQ eg: DCQ 0X12345678FF0055AA.
5. Floating point single precision.	DCFS eg: DCFS 1E3.
6. Floating point Double precision.	DCFD eg: DCFD 3.14159.
7. Instruction,	DCI eg: DCI 0XBE00; Breakpoint (BKPT 0).

unified Assembly language (UAL):

A number of years age, The pre-UAL assembly code syntax used were less explicit and the omissions of 'S' suffixes in many data processing instructions were allowed.

As the ARM architecture evolved 32-bit thumb② instructions are introduced with Thumb-2 technology and the ambiguity of legacy Syntax become a problem because many thumb instructions have the option of of updating the APSR (or) not updating the APSR. The UAL syntax was developed to solve this issue; as well as allowing consistent syntax for both Thumb and ARM assembly codes.

<u>differences of preUAL & UAL</u>:

→ some data operation instructions use three operands even when the destination register is the same as one of the source registers. while in the past (pre-UAL) syntax might ~~not~~ only use two operands for the same instruction.

→ The 'S' suffix becomes more explicit. In the past, when an assembly program file is assembled into thumb code, most data operations are implied as instructions that update APSR, as a result, the 'S' suffix is not essential. with UAL syntax, instructions that update APSR should has 'S' suffix to clearly indicate the expected operation.

ex:

pre-UAL: MOV R0, R1 [R1 → R0, update APSR]

UAL: MOVS R0, R1 [R1 → R0, update APSR].

Instruction List.

The instructions in the cortex-M0 processors can be devided into various groups based on functionality.

1. Moving data with in the processor.
2. Memory access.
3. stack-Memory access.
4. Arithmatic operations.
5. logic operations.
6. shift & rotate operations.
7. extend and reverse ordering operations.
8. program flow control.
9. Memory barrier instructions.
10. Exception-Related instructions.
11. other functions.

(1) Moving data with in the processor:

(1)

Instruction	MOV
Function	Move register into register.
UAL :	MOV <Rd>, <Rm>
pre UAL :	MOV <Rd>, <Rm> (or) CPY <Rd>, <Rm>.

(2)

Instruction	MOVS
Function	Move register into register and update APSR.
UAL :	MOVS <Rd>, <Rm>
pre-UAL :	MOVS <Rd>, <Rm>.

③ Instruction MOV

Function Move Immediate data (# sign extension) into register.

UAL MOVS <Rd>, # immed8.

pre-UAL MOV <Rd>, # immed8.

{Immediate data range 0 to +255, APSR.Z and APSR.N update}.

4. Spectial register accesses : CONTROL, PRIMASK, xPSR.

Instruction MRS

Function Move special register into register.

MRS <Rd>, <special Reg>

Ex:- MRS R0, CONTROL
MRS R1, PRIMASK.
MRS R3, xPSR.

Instruction MSR

Function Move register into special register.

MSR <special Reg>, <Rd>

Ex:- MSR CONTROL, R1.
MSR PRIMASK, R0.

Stack memory Access :-

The PUSH & POP instructions are dedicated to stack memory access. The PUSH instruction is used to decrement the current SP and store data to the stack. The POP instruction is used to read the data from the stack and increment the current SP.

Instruction PUSH

Function write single (or) multiple registers into memory and update base register (SP).

Syntax PUSH {<Ra>, <Rb> ...}

PUSH {<Ra>, <Rb> ...., LR}.

note:- new_sp = SP-4 X no.of register to PUSH.

Instruction POP

Function Read single (or) multiple registers from memory and update base register (SP).

syntax POP {<Ra>, <Rb> ....}

POP {<Ra>, <Rb>, ---, PC}.

Memory Access Instructions

Various Transfer Sizes

Size.	unsigned. load	signed load	signed/unsigned store.
word	LDR	LDR	STR
Half word	LDRH	LDRSH	STRH.
Byte.	LDRB	LDRSB	STRB.

(i) memory read operation, the instruction to carry out single access is LDR :

Instruction. LDR/LDRH/LDRB

Function Read single memory data into register.

LDR <Rt>, [<Rn>, <Rm>]

LDRH <Rt>, [<Rn>, <Rm>]

LDRB <Rt>, [<Rn>, <Rm>]

Rt = memory [Rn + Rm]

Rt, Rn, Rm are low registers.

(ii) Cortex-M processor supports immediate offset addressing modes:

instruction   LDR | LDRH | LDRB

Function   Read single memory data into register.

```
LDR   <Rt>, [<Rn>, #immed5]
LDRH  <Rt>, [<Rn>, #immed5]
LDRB  <Rt>, [<Rn>, #imm5]
```

Rt = memory [Rn + zero extent (#immed5 <<2)] ; word.
= " [" " " <<1] , Halfword
= " [" " "] Byte.

(iii) The cortex-M processor support a useful PC/SP relative load instruction allowing efficient literal data accesses.

instruction.   LDR.

Function   Read single memory data word into register.

```
LDR <Rt>, [PC, #immed8]
```

```
Ex: LDR R0, =0X12345678
    LDR R0, [PC, #0X40]
    LDR R0, Label.
```

```
LDR <Rt>, [SP, #immed8]
```

```
Ex:- LDR R0, [SP, #0X20].
```

Instruction LDM (Load multiple)

function: Read multiple memory data word into registers, base address register update by memory Read.

LDM <Rn>, {<Ra>, <Rb>...}

Ra = memory[Rn]
Rb = memory [Rn+4].

Instruction LDMIA / LDMFD.

Load Increment After / Base address register update to sussequence ~~address~~ addrss.

function: Read multiple memory data word into registers and update base register.

LDMIA <Rn>!, {<Ra>, <Rb>...}

Ra = memory [Rn];
Rb = memory [Rn+4].

Ex: LDMIA R0!, {R1, R2, R5, R7}.

Read multiplple registers, R0 update to address after last read operation.

LDMFD is another name for the same instruction, which was used for restoring data from a Full descending stack, in traditional ARM systems that use software managed stack.

(iii) Sign extended load & instructions.

instruction LDRSB | LDRSH

Function Read ~~sign~~ single signed memory data into register.

LDRSH <Rt>, [<Rn>, <Rm>]
LDRSB <Rt>, [<Rn>, <Rm>].

store instructions

instruction STR| STRH| STRB

Function write single register data into memory.

STR <Rt>, [Rn>, <Rm>]
STRH <Rt>, [<Rn>,<Rm>]
STRB <Rt>, [Rn>, <Rm>].

Function STR| STRH| STRB

write single memory data into memory.
with immediate offset addressing.

STR <Rt>, [<Rn>, *immed5]
STRH <Rt>, [" "]
STRB <Rt>, [" "]

instruction STR

function: write single memory data word in to memory. [An SP relative store instruction which supports a wider offset range].

STR <Rt>, [SP, *immed8],

Rt = memory[SP+ zero extend (*immed8 <<2].

instruction : STMIA (store multiple increment after)
STMEA

function   write multiple register data into memory
and update base register.

STMIA <Rn>!, {<Ra>, <Rb>...}

Arithmatic operations

instruction   ADD

Add two registers.

(1)   UAL   ADDS <Rd>, <Rn>, <Rm>.
pre-UAL   ADD <Rd>, <Rn>, <Rm>.

Rd = Rn + Rm, APSR update

Add an immediate constant into a register

(2)   UAL   ADDS <Rd>, <Rn>, #immd3.
pre-UAL   ADD <Rd>, <Rn>, #imed3.

(3) Add two registers without updating APSR.

UAL: ADD <Rd>, <Rm>

Rd = Rd + Rm

(4) Add stack pointer to a register without updating APSR.

UAL: ADD SP, <Rm>
pre-UAL: ADD SP, <Rm>   SP = SP + Rm

→ Add stack pointer to a Register with out updating APSR:

UAL: ADD <Rd>, Sp, #immd8.

ADD <Rd>, SP, #immd 8

Rd = SP ~~+~~zero entend (#immed8 << 2).

→ Add an immediate constant with program counter to a register with out updating APSR.

ADR <Rd>, <label>.

ADR <Rd>, PC, #immd8

→ Add with carry and update APSR

UAL: ADCS <Rd>, <Rm>.

pre UAL: ADC <Rd>, <Rm>.

Rd = Rd + Rm + carry

→ subtract two registers

UAL: SUBS <Rd>, <Rn>, <Rm>.

pre-UAL: SUB <Rd>, <Rn>, <Rm>

→ substract a register with an immediate constant.

UAL: SUBS <Rd>, <Rn>, #immd3

pre-UAL: SUB <Rd>, <Rn>, #immed 3

→ subtract with Barrow.

UAL: SBCS <Rd>, <Rd>, <Rm>

pre-UAL: SBC <Rd>, <Rm>

Rd = Rd - Rm - Borrow, APSR update

→ Reverse subtract (negative).

UAL: RSBS <Rd>, <Rn>, #0.

pre-UAL: NEG <Rd>, <Rm>.

→ Simple multiplication.

UAL: MULS <Rd>, <Rm>, <Rd>.

MUL <Rd>, <Rm>.

Rd = Rd * Rm, APSR.N & APSR.Z update.

→ compare (using subtract) values and update flags APSR, but the result of the compare is not stored.

UAL: CMP <Rn>, <Rm>.

Calculate Rn − Rm, APSR update but subtract result is not stored.

→ compare negative.

UAL: CMN <Rn>, <Rm>.

Calculate Rn − Neg (Rm), APSR update.

⇒ ~~logic instruction~~ logic operations

→ logical AND

UAL: ANDS <Rd>, <Rd>, <Rm>

pre-UAL: AND <Rd>, <Rm>

Rd = AND (Rd, Rm), APSR.N & APSR.Z update.

→ logical OR.

UAL: ORRS <Rd>, <Rd>, <Rm>.

pre_UAL: ORR <Rd>,<Rm>.

→ logical exclusive OR-operation

UAL: EORS <Rd>, <Rd>, <Rm>.

EOR <Rd>, <Rm>

→ logic Bitwise clear

UAL: BICS <Rd>, <Rd>, <Rm>.

preUAL: BIC <Rd>,<Rm>.

→ logic Bitwise not operation.

UAL: MVNS <Rd>,<Rm>

MVN <Rd>, <Rm>.

Rd = NOT (Rm>

→ logic Test (~~bit AND~~ bitwise AND).

UAL: TST <Rn>, <Rm>.

calculate AND (Rn, Rm), update APSR.N, & APSR.Z

but result not stored.

shift & rotate operations:

1. instruction ASR

Arithmatic shift right.

UAL: ASRS <Rd>, <Rd>, <Rm>

pre-UAL: ASR <Rd>, <Rm>.

LSL

logical shift left.

UAL: LSLS <Rd>, <Rd>, <Rm>

pre-UAL: LSL <Rd>, <Rm>.

LSR

logical shift right

UAL: LSRS <Rd>, <Rd>, <Rm>

pre-UAL: LSR <Rd>, <Rm>.

instruction ROR.

Rotate Right.

UAL: RORS <Rd>, <Rd>, <Rm>.

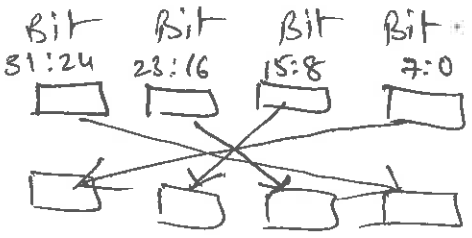
ROR <Rd>, <Rm>.

Extend and reverse ordering Operations:

instruction: REV.

Byte order Reverse.

REV <Rd>, <Rm>.

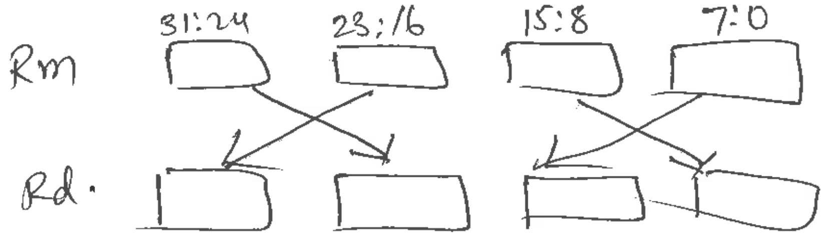


instruction: REV16

Byte order Reverse with in halfword.

REV16 <Rd> <Rm>.

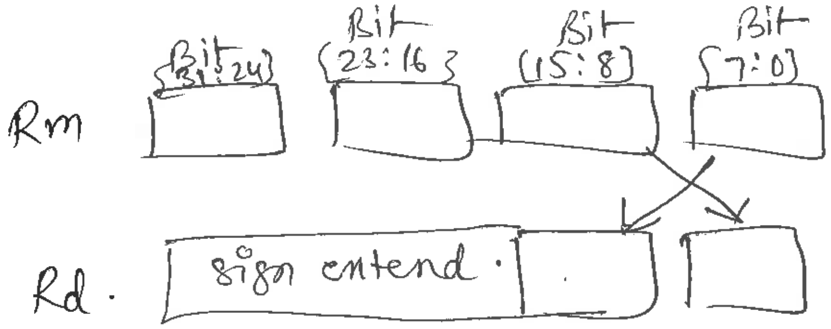
Rd = { Rm[23:16], Rm[31:24], Rm[7:0], Rm[15:8] }.



instruction: REVH

Byte order reverse with in lower half word, then sign extend result.

REVSH <Rd>, <Rm>.



extending operations

instruction: SXTB

sign extend lowest byte in a word of data

SXTB <Rd>,<Rm>.

Ex: Rd = sign extend(Rm[7:0])

instruction: SXTH

sign extend lower half word in a word of data

SXTH <Rd>, <Rm>

unsigned extend Byte:

instruction UXTB

extend lowest byte in a word of data.

UXTB <Rd>, <Rm>

Rd = zero extend (Rm [7:0])

instruction UXTH

unsigned extended Half word.

Extend lower half word in a word of data.

UXTH <Rd>, <Rm>

Rd = zeroextend (Rm [15:0])

program flow control instructions

instruction B (Branch)

Branch to an address (unconditional)

syntax: B <label>

Branch range is ± 2046 bytes of current program counter.

instruction: ~~syntax~~: B <cond> (conditiond)

Depending of APSR, branch to an address.

Syntax: B <cond> <label>

Branch range is ± 254 bytes of current program counter.

Syntax: BL <label>

Branch range is ± 16MB of current program counter.

Syntax: BX <Rm>

Branch to an address specified by a register, and change processor state depending on bit [0] of the register.

Syntax: BLX <Rm>

Branch to an address specified by a register, save return address to link register and change processor state depending of bit [0] of the register.

Conditional Suffixes for conditional branches

1.	EQ	Equal	ZF = 1
2.	NE	Not equal	ZF = 0
3.	CS/HS	carry set	C = 1
4.	CC/LO	carry clear	C = 0
5.	MI	Minus / Negative	N = 1
6.	PL	Plus / positive (or) zero	N = 0
7.	VS	overflow	OV = 1
8.	VC	NO overflow	OV = 0.
9	HI	unsigned higher	C = 1, & Z = 0
10.	LS	unsigned lower	C = 0, & Z = 1
11	GE	Greater than (or) Equal	N = 1, V = 1. (or) N = 0 & V = 0.
12	LT	Less than	
13.	GT	greater than	

Memory Barrier Instructions

1. Instruction DMB
   Data Memory Barrier.

   Syntax: DMB.
   Ensures that all memory accesses are completed before new memory access is committed.

2. Instruction DSB
   Data synchronization Barrier.

   Syntax: DSB
   Ensures that all memory accesses are completed before next instruction is executed

3. Instruction: ISB
   Instruction Synchronization Barrier.

   Syntax: ISB
   Flushes the pipe line and ensure that all previous instructions are completed before executing new instructions.

Exception Instructions

Instruction SVC
supervisor call.

Syntax: SVC #Immed8>.

SVC #3, Trigger the SVC instruction.

Instruction CPS

change processor state: enable(or) disable interrupt.

Syntax: CPSIE I ; enable interrupt.

CPSID I ; disable interrupt.

Sleep mode ~~feat~~ feature Related Instructions

Instruction: WFI

wait for interrupt.

syntax: WFI

stops program execution untill an interrupt arrived, (or) if the processor entered a debug state.

Instruction WFE

wait for event.

Syntax: WFE

if an interal event register is set. It clears the internal event register and continues execution.

Instruction SEV

send event to all processors in multiprocessing environment.

syntax: SEV

Set local event register and send out an event pulse to other microprocessor in a multiple processor system.

Other Instructions

① Instruction NOP

No operation.

NOP

② Instruction BKPT

Break point.

BKPT #<immed8>

BKPT instruction can have an 8-bit immediate data. This can be used by the debugger as an identifier for the BKPT. ~~for~~