

Internal Assessment Test 1

Solution

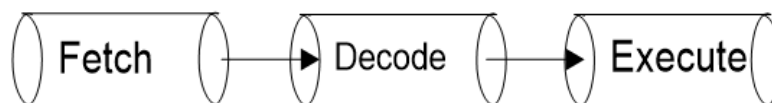
Sub:	MICROCONTROLLERS					Sub Code:	BCS402	Branch:	CSE
Date:	27/04/2025	Duration:	90 mins	Max Marks:	50	Sem / Sec:	4 A,B,C		

1.a

Describe pipeline in ARM. Illustrate with an example pipeline stage of ARM 7 & ARM 9. Pipeline .

Pipeline is the mechanism to speed up execution by fetching the next instruction while other instructions are being decoded and executed.

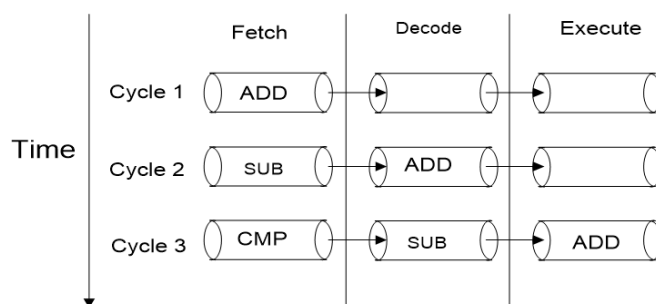
Figure shows the ARM7 three-stage pipeline.



Fetch loads an instruction from memory.

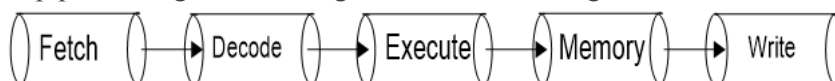
Decode identifies the instruction to be executed.

Execute processes the instruction and writes the result back to a register.



The ADD instruction is executed, the SUB instruction is decoded, and the CMP instruction is fetched. This procedure is called filling the pipeline.

The pipeline design for each ARM family differs. For example, the ARM9 core increases the pipeline length to five stages as shown in the figure below.



1.b

PRE-Condition:

R5=5

R7=8

MOV R7, R5, LSL #2

After execution of the above instructions what will be the value of R5 and R7.

Instruction Execution:

1. MOV R7, R5, LSL #2

- This instruction moves the value of R5 into R7, but with a Left Shift Logical (LSL) by 2 bits.
- Left shifting by 2 bits is equivalent to multiplying by $2^2=4$

2. $R7=R5 \times 4$

3. $R7 = 5 \times 4 = 20$

Final Values After Execution:

- R5 remains unchanged $\rightarrow R5 = 5$
- R7 is updated to 20 $\rightarrow R7 = 20$

Given Pre-Condition:

- $R5=5$
 $R5 = 5$
 $R5 = 5 \rightarrow$ Binary: 0000 0000 0000 0000 0000 0000 0000 0101
- $R7=8$
 $R7 = 8$
 $R7 = 8 \rightarrow$ Binary: 0000 0000 0000 0000 0000 0000 0000 1000

Instruction Execution:

MOV R7, R5, LSL #2

- This instruction **left shifts** the value in R5 by 2 bits.

R5 before shifting (5 in binary):

0000 0000 0000 0000 0000 0000 0000 0101

Left Shift by 2 bits (LSL #2):

0000 0000 0000 0000 0000 0000 0001 0100

The decimal equivalent of 0000 0000 0000 0000 0000 0000 0001 0100 is

20.

2.a

Explain the ARM Core data flow model with a neat diagram

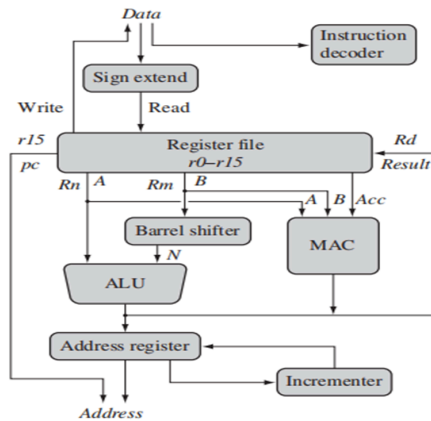


Figure 2.1 ARM core dataflow model.

- ARM core as functional units connected by data buses.
- The arrows represent the flow of data, the lines represent the buses, and the boxes represent either an operation unit or a storage area.
- Data enters the processor core through the Data bus. The data may be an instruction to execute or a data item.

The instruction decoder translates instructions before they are executed.

The ARM processor, like all RISC processors, uses a load - store architecture.

Load instructions copy data from memory to registers, and conversely the store instructions copy data from registers to memory.

ARM instructions typically have two source registers, Rn and Rm, and a single destination register, Rd.

Source operands are read from the register file using the internal buses A and B, respectively.

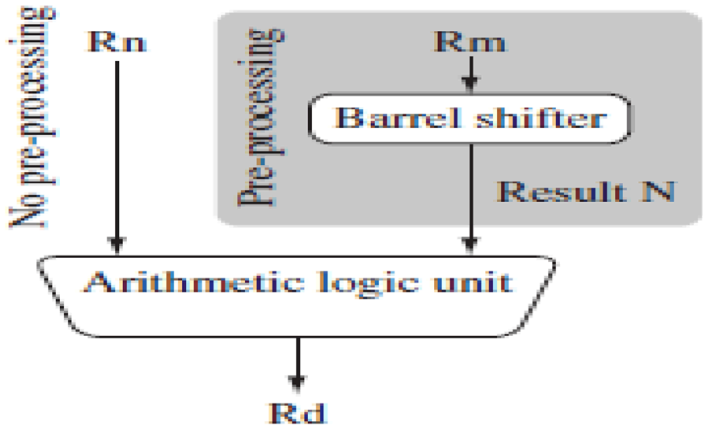
The ALU (arithmetic logic unit) or MAC (multiply-accumulate unit) takes the register values Rn and Rm from the A and B buses and computes a result.

Load and store instructions use the ALU to generate an address to be held in the address register and broadcast on the Address bus.

One important feature of the ARM is that register Rm alternatively can be preprocessed in the barrel shifter before it enters the ALU.

After passing through the functional units, the result in Rd is written back to the register file using the Result bus.

For load and store instructions the incrementer updates the address register.

2.b	<p>Illustrate the working of forward and backward branches with the help of suitable examples</p> <p>The example shown below is a forward branch. The forward branch skips three instructions.</p> <pre> B forward ADD r1, r2, #4 ADD r0, r6, #2 ADD r3, r7, #4 forward SUB r1, r2, #4 </pre> <p>The backward branch creates an infinite loop.</p> <pre> backward ADD r1, r2, #4 SUB r1, r2, #4 ADD r4, r6, r7 B backward </pre>
3.a	<p>Explain the Barrel shifter Operation in the ARM processor, with a neat diagram.</p>  <pre> graph TD Rn[Rn] -- "No pre-processing" --> ALU[Arithmetic logic unit] Rm[Rm] -- "Pre-processing" --> BS[Barrel shifter] BS -- "Result N" --> ALU ALU --> Rd[Rd] </pre>

● A unique feature of the ARM processor is the ability to shift the 32-bit binary pattern in one of the source registers left or right by a specific number of positions before it enters the ALU.

● This shift increases the power and flexibility of many data processing operations.

● For example:

PRE r5=5

r7=8

MOV r7, r5, LSL #2

POST r5=5

r7=20

● The example multiplies register r5 by four and then places the result into register r7.

This is particularly useful for loading constants into a register and achieving fast multiplies or division by a power of 2.

Barrel shifter operations.

Mnemonic	Description	Shift
LSL	logical shift left	xLSL y
LSR	logical shift right	xLSR y
ASR	arithmetic right shift	xASR y
ROR	rotate right	xROR y
RRX	rotate right extended	xRRX

Note: x represents the register being shifted and y repres

Barrel shift operation syntax for data processing instructions.

N shift operations	Syntax
Immediate	#immediate
Register	Rm
Logical shift left by immediate	Rm, LSL #shift_imm
Logical shift left by register	Rm, LSL Rs
Logical shift right by immediate	Rm, LSR #shift_imm
Logical shift right with register	Rm, LSR Rs
Arithmetic shift right by immediate	Rm, ASR #shift_imm
Arithmetic shift right by register	Rm, ASR Rs
Rotate right by immediate	Rm, ROR #shift_imm
Rotate right by register	Rm, ROR Rs
Rotate right with extend	Rm, RRX

PRE Condition:

r0 = 0x00000000

r1 = 0x00009000

mem32[0x00009000] = 0x01010101

mem32[0x00009004] = 0x02020202

Write the POST Conditions for the following single register addressing modes:

1) LDR r0, [r1, #4]!

Pre Indexing with Writeback:

	<pre> PRE r0 = 0x00000000 r1 = 0x00090000 mem32[0x00009000] = 0x01010101 mem32[0x00009004] = 0x02020202 LDR r0, [r1, #4]! Preindexing with writeback: POST(1) r0 = 0x02020202 r1 = 0x00009004 LDR r0, [r1, #4] Preindexing: POST(2) r0 = 0x02020202 r1 = 0x00009000 LDR r0, [r1], #4 Postindexing: POST(3) r0 = 0x01010101 r1 = 0x00009004 </pre>
	<p>Explain the different processor modes provided by ARM7.</p> <p>Each processor mode is either privileged or nonprivileged.</p> <p>A privileged mode allows read-write access to the cpsr.</p> <p>A nonprivileged mode only allows read access to the control field in the cpsr but allows read-write access to the conditional flags.</p> <p>There are seven processor modes : six privileged modes and one nonprivileged mode.</p> <p>The privilege modes are abort, fast interrupt request, interrupt request, supervisor, system and undefined.</p> <p>The nonprivileged mode is user.</p> <p><u>Processor Modes:</u></p>

- 1.The processor enter abort mode when there is a failure to attempt to access memory.
- 2.Fast interrupt request and interrupt request modes correspond to the two interrupt levels available on the ARM processor.
- 3.Supervisor mode is the mode that the processor is in after reset and is the mode that an operating system kernel operates in.
- 4.System mode is a special version of user mode that allows full read-write access to the cpsr.
- 5.Undefined mode is used when the processor encounters an instruction that is undefined or not supported by the implementation.
- 6.User mode is used for program and applications .

Write an ALP to multiply two 16-bit binary numbers

```
AREA SUM,CODE,READONLY
```

```
ENTRY
```

```
LDR R0,=NUM
```

```
LDRH R1,[R0]
```

```
LDRH R2,[R0,#4]
```

```
MUL R3,R1,R2
```

```
STOP B STOP
```

```
NUM DCD 0x00000006, 0x00000006
```

```
END
```


5

With an example explain the following ARM instructions.

i) MOV

● It copies N into a destination register Rd, where N is a register or immediate value.

● This instruction is useful for setting initial values and transferring data between registers.

Syntax: <instruction> {<cond>} {S} Rd, N

ii) MVN

MOV	Move a 32-bit value into a register	$Rd = N$
MVN	move the NOT of the 32-bit value into a register	$Rd = \sim N$

iii) ADC

ADC	add two 32-bit values and carry	$Rd = Rn + N + \text{carry}$
-----	---------------------------------	------------------------------

iv) RSC

RSC	reverse subtract with carry of two 32-bit values	$Rd = N - Rn - !(\text{carry flag})$
-----	--	--------------------------------------

v) BIC

BIC	logical bit clear (AND NOT)	$Rd = Rn \& \sim N$
-----	-----------------------------	---------------------

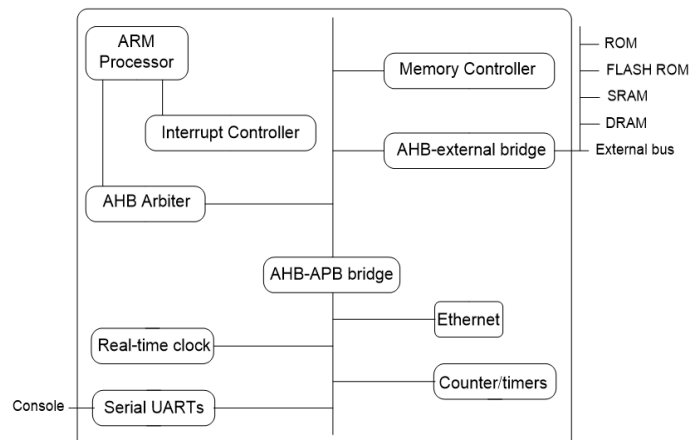
6.a

With a neat diagram explain the four main hardware components of an ARM based embedded device.

ARM processor based embedded system hardware can be separated into the following four main hardware components:

◦ The ARM processor: The ARM processor controls the embedded device. Different versions of the ARM processor are available to suit the desired operating characteristics.

- Controllers: Controllers coordinate important blocks of the system. Two commonly found controllers are memory controller and interrupt controller.
- Peripherals: The peripherals provide all the input-output capability external to the chip and responsible for the uniqueness of the embedded device.
- Bus: A bus is used to communicate between different parts of the device.
- The ARM processor core is a bus master—a logical device capable of initiating a data transfer with another device across the same bus.
- Peripherals tend to be bus slaves—logical devices capable only of responding to a transfer request from a bus master device.
- The Advanced Microcontroller Bus Architecture (AMBA) was introduced in 1996 and has been widely adopted as the on-chip bus architecture used for ARM processors.
- The first AMBA buses introduced the ARM System Bus (ASB) and the ARM Peripheral Bus (APB).
- Later ARM introduced another bus design, called the ARM High Performance Bus (AHB).
- AHB provides higher data throughput than ASB because it is based on a centralized multiplexed bus scheme.



6.b

Compare Microprocessors and microcontrollers.

Microprocessor	Micro Controller
Microprocessor is heart of Computer system.	Micro Controller is a heart of embedded system.
It is just a processor. Memory and I/O components have to be connected externally	Micro controller has external processor along with internal memory and i/O components
Since memory and I/O has to be connected externally, the circuit becomes large.	Since memory and I/O are present internally, the circuit is small.
Cannot be used in compact systems and hence inefficient	Can be used in compact systems and hence it is an efficient technique
Cost of the entire system increases	Cost of the entire system is low
Due to external components, the entire power consumption is high. Hence it is not suitable to use with devices running on stored power like batteries.	Since external components are low, total power consumption is less and can be used with devices running on stored power like batteries.
Most of the microprocessors do not have power saving features.	Most of the micro controllers have power saving modes like idle mode and power saving mode. This helps to reduce power consumption even further.
Since memory and I/O components are all external, each instruction will need external operation, hence it is relatively slower.	Since components are internal, most of the operations are internal instruction, hence speed is fast.
Microprocessor have less number of registers, hence more operations are memory based.	Micro controller have more number of registers, hence the programs are easier to write.