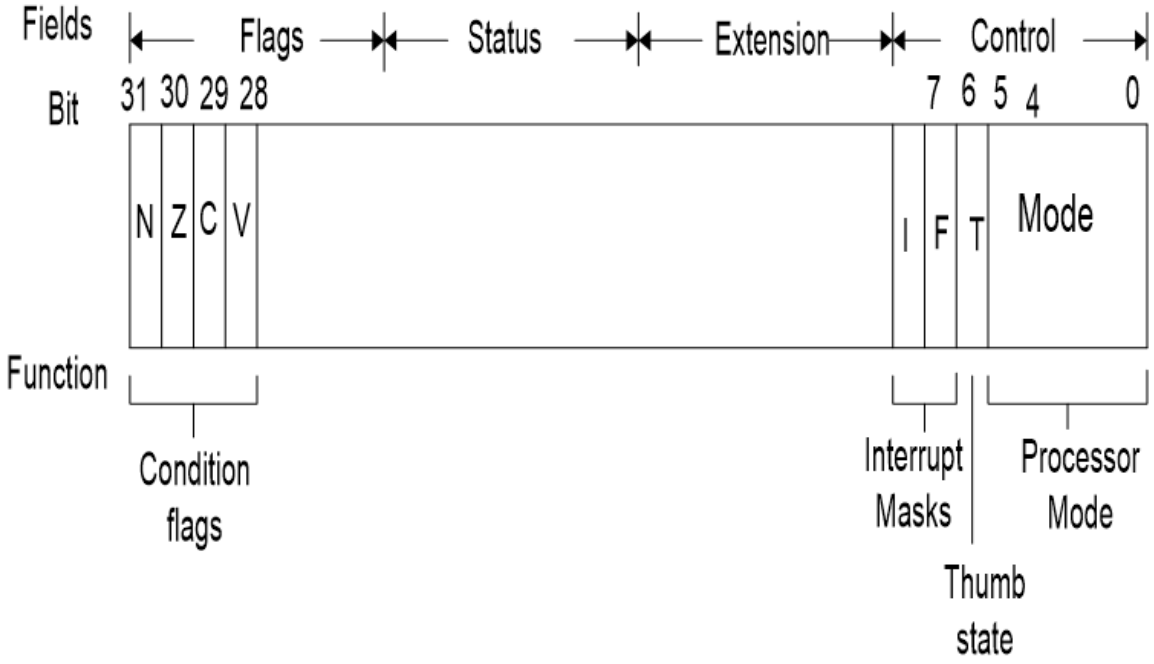
Q1A. Draw the layout of a generic program status register and briefly explain the various fields. **[6 MARKS]**

ANS:



The cpsr is divided into four fields, each 8 bits wide: flags, status, extension and control.

In current designs the extension and status fields are reserved for future use.

The control field contains the processor mode, state and interrupts mask bits.

The flag field contains the condition flags.

When cpsr bit 5, T=1, then the processor is in Thumb state. When T=0, the processor is in ARM state.

Each processor mode is either privileged or non-privileged.

A privileged mode allows read-write access to the cpsr.

A non-privileged mode only allows read access to the control field in the cpsr but allows read-write access to the conditional flags.

There are seven processor modes : six privileged modes and one non-privileged mode.

The privilege modes are abort, fast interrupt request, interrupt request, supervisor, system and undefined.

The non-privileged mode is user.

The cpsr has two interrupt mask bits, 7 and 6 (I and F) which control the masking Interrupt request (IRQ) and Fast Interrupt Request (FIR).

if a SUBS subtract instruction results in a register value of zero, then the Z flag in the cpsr is set.

Q 1B. What are the banked registers ? Explain briefly. **[4 MARKS]**

Figure shows all 37 registers in the register file.
Of these, 20 registers are hidden from a program at different times. These registers are called banked registers & are identified by the shading in the diagram.
They are available only when the processor is in a particular mode, for example, abort mode has banked registers r13_abt, r14_abt and spsr_abt.
Banked registers of a particular mode are denoted by an underline character post-fixed to the mode mnemonic or _mode.
Every processor mode except user mode can change mode by writing directly to the mode bits of the cpsr.
All privileged modes except system mode have a set of associated banked registers that are subset of the main 16 registers.
If the processor mode is changed, a banked register from the new mode will replace an existing register.
Exceptions and interrupts suspend the normal execution of sequential instructions and jump to a specific location.
The following exception and interrupts causes a mode change:
◦Reset, interrupt request, fast interrupt request, software interrupt, data abort, prefetch abort and undefined instructions.

Q2A . Explain the ARM Core data flow model with a neat diagram.
[6 MARKS]

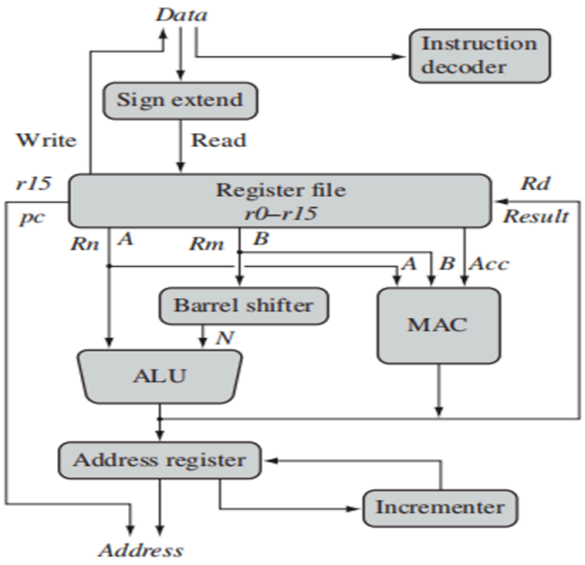


Figure 2.1 ARM core dataflow model.

•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.

Q2B. Does ARM is based on pure RISC architecture? Justify your answer. **[4 MARKS]**

RISC design philosophy
◦The ARM core uses a RISC architecture.
◦Aimed at delivering simple but powerful instructions that execute within a single cycle at a high clock speed.
◦Concentrates on reducing the complexity of instructions performed by the hardware.
◦Provides greater flexibility and intelligence in software rather than hardware.

Q 3.A
PRE-Condition: R0 = 0x00000000 , R1= 0x00006000 mem32[0x00006000] = 0x05040403 mem32[0x00006004]0x06040804 Find the POST Conditions for the following cases: LDR R0, [R1, #4]! ii) LDR R0, [R1, #4] iii) LDR R0, [R1], #4

[6 MARKS]

i) LDR R0, [R1, #4]!:

- The instruction loads mem32[0x00006004] (0x06040804) into R0.
- R1 is incremented by 4.
- So, after this instruction:
    - R0 = 0x06040804
    - R1 = 0x00006004

ii) LDR R0, [R1, #4]:

- The instruction loads mem32[0x00006004] (0x06040804) into R0.
- R1 remains unchanged.
- So, after this instruction:
    - R0 = 0x06040804
    - R1 = 0x00006000 (unchanged)

iii) LDR R0, [R1], #4:

- The instruction loads mem32[0x00006000] (0x05040403) into R0.
- R1 is incremented by 4.
- So, after this instruction:
    - R0 = 0x05040403
    - R1 = 0x00006004

So, the post-conditions for each case are:

i) LDR R0, [R1, #4]!: R0 = 0x06040804, R1 = 0x00006004 ii) LDR R0, [R1, #4]: R0 = 0x06040804, R1 = 0x00006000 iii) LDR R0, [R1], #4: R0 = 0x05040403, R1 = 0x00006004

Q 3B.

PRE-Condition: R0=0x00000000 , R1=0x00000005 After execution of the below instruction what will be the value of R0 and R1?(mention all the steps) ADD R0,R1,R1,LSL #1

[4 MARKS]

1. **Initial Conditions**:

   - R0 = 0x00000000

   - R1 = 0x00000005

2. **Execution**:

   - `LSL #1`: Left shift R1 by 1 bit using a barrel shifter.

     - Result: R1 = 0x0000000A

   - `ADD R0, R1, R1`: Add R1 to itself.

     - Result: R0 = R1 + R1 = 0x00000005 + 0x0000000A = 0x00000015 =F

3. **Post-conditions**:

   - After the execution of the instruction `ADD R0, R1, R1, LSL #1`:

     - R0 = 0x00000015

     - R1 = 0x0000000A

Q 4 A.

PRE-Condition: r0=0x00000000, r1= 0x00000000, r2= 0xf0000002, r3=0x00000002.

Find the post condition for the following instructionUMULL r0,r1, r2, r3

[5 MARKS]

1. **Initial Conditions**:

   - r0 = 0x00000000

   - r1 = 0x00000000

   - r2 = 0xF0000002

   - r3 = 0x00000002

2. **Execution**:

   - Unsigned multiplication of r2 and r3:

     - Result: 0xF0000002 * 0x00000002 = 0x1E00000004

     - Store lower 32 bits in r0: r0 = 0x00000004

     - Store upper 32 bits in r1: r1 = 0x1E000000

3. **Post-conditions**:

   - After the execution of the instruction `UMULL r0, r1, r2, r3`:

     - r0 = 0x00000004

     - r1 = 0x1E000000

     - r2 and r3 remain unchanged

So, the post-conditions are:

- r0 = 0x00000004

- r1 = 0x1E000000

- r2 = 0xF0000002

- r3 = 0x00000002

Q 4B. Write an ALP using ARM instructions to multiply two 16-bit numbers and store the result in memory. **[5 MARKS]**

```
1 PROGRAM

AREA ARTH, CODE, READONLY

ENTRY

LDR R0, =NUM

LDRH R1, [R0]

LDRH R2, [R0, #2]

MUL R3, R1, R2

STOP B STOP

NUM DCW 0X0006, 0X0006

END
```

OR

```
2ND PROGRAM

MOV R1, #0X0006

MOV R2, #0X0006

MUL R3,R1,R2

LDR R0,=RESULT

STR R3, [R0]
```

```
AREA DATA, DTA, READWRITE

RESULT DCD 0X0

END
```

Q 5 A. Write an ALP using ARM instructions for the given statement If (r2! =10) , r5=(r5+r2)-r3 [5 MARKS]

```
    AREA if_statement, CODE, READONLY

    ENTRY

    ; Compare r2 with 10

    CMP   r2, #10       ; Compare r2 with 10

    BEQ   skip_calculation ; Branch if equal to 10

    ; If r2 is not equal to 10, perform the calculation

    ADD   r5, r5, r2    ; Add r2 to r5

    SUB   r5, r5, r3    ; Subtract r3 from r5

skip_calculation

    ; Place the rest of your code here

    END
```

In this code:

- We compare the value in register r2 with the immediate value 10.

- If r2 is not equal to 10, we execute the calculation: `r5 = (r5 + r2) - r3`.

- If r2 is equal to 10, we skip the calculation and continue with the rest of the code.

Q 5B. Refer the following snippet and find the address hold by program counter and link register before execution of subroutine and after execution of subroutine. **[5 MARKS]**

Address 0x00000000 BL subroutine

0x00000004 CMP r1,#5

0x00000008 MOVEQ r1, #0

:

:

Subroutine 0x00000020 Addr1,#2

0x00000030 Mov pc, lr

ANS:

To analyze the addresses held by the Program Counter (PC) and Link Register (LR) before and after the execution of the subroutine, let's follow the execution flow step by step:

Before Execution of Subroutine:

- Address: 0x00000000

  - The instruction `BL subroutine` is executed, which performs a branch with link to subroutine.

  - PC (Program Counter) holds the address of the next instruction after the BL instruction, which is 0x00000004.

  - LR (Link Register) holds the address of the instruction following the BL instruction, which is 0x00000004.

After Execution of Subroutine:

- Address: 0x00000020 (Inside the subroutine)

  - The subroutine is executed.

  - The instruction `Mov pc, lr` is executed, which moves the value of LR into PC, effectively returning control to the calling function.

- PC holds the value stored in LR, which is 0x00000004 (the address following the BL instruction).

- LR holds the return address of the subroutine, which is 0x00000008 (the address following the BL instruction in the main code).

So, to summarize:

- Before the subroutine execution:

  - PC holds: 0x00000004

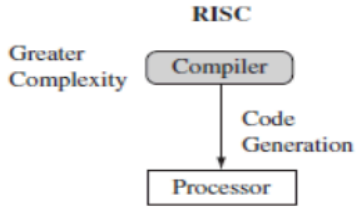
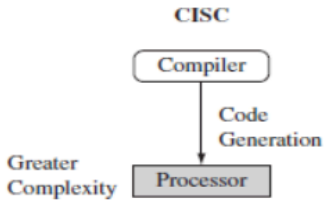
  - LR holds: 0x00000004

- After the subroutine execution:

  - PC holds: 0x00000004

  - LR holds: 0x00000008

Q 6 A. Differentiate CISC and RISC architectures and explain four major rules of RISC design.**[7 MARKS]**

RISC	CISC
	
Emphasizes on compiler complexity	Emphasizes on processor complexity
Simple but powerful instructions	Instructions are more complicated
Executes instruction in single cycle	Takes many cycle to execute

Instructions are of fixed length	Instructions are of variable length
Have large set of general purpose registers	Have limited set of general purpose registers
Any register can contain either data or an address	Dedicated registers for specific purpose
Separate load and store instructions transfer data between the register and external memory.	MOV instructions can be used to transfer between register and memory.

Q6B. Compare microprocessors and microcontrollers.[3 MARKS]

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 used 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.