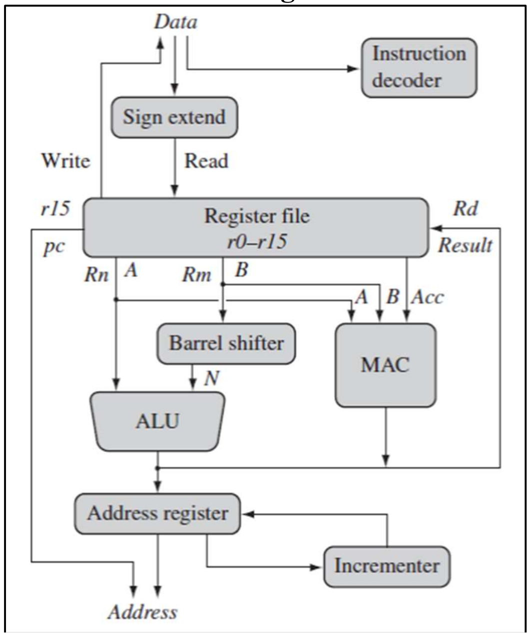
1. **Compare and Contrast microprocessor and microcontroller.**

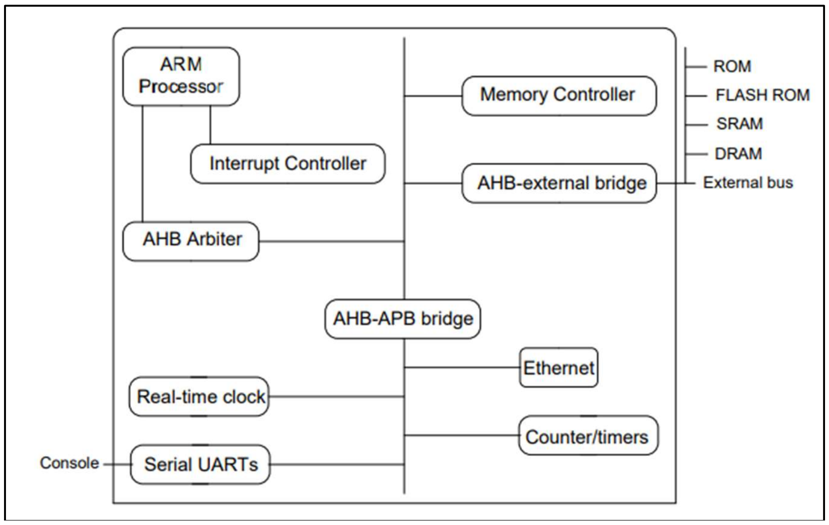
| **Microprocessor** | **Microcontroller** |
|---|---|
| It is an electronic component that is used by a computer to do work. It is a CPU on a single IC chip containing millions of transistors, registers and diodes that work together. | It is a compact IC chip designed to monitor a specific operation in an embedded system. A typical microcontroller includes a processor, memory, and I/O peripherals on a single chip. |
| Generally, does not have RAM, ROM, and I/O pins | Is 'all-in-one'. Processor with RAM, ROM, and I/O pins, all in one chip. |
| Usually uses its pins as the bus to interface the RAM, ROM, and peripherals. Hence the controlling bus is expandable at the board level. | Controlling bus is internal and not available to the board design. |
| Generally capable of being built into bigger general-purpose applications. | Usually used for more dedicated applications. |
| Generally, do not have power saving systems | Have power saving system like idle mode or power saving mode. |
| Overall cost of system made with microprocessors are high, because of the high number of external components required. | Made using CMOS technology so they are cheaper than microprocessors. |
| Processing speed is above 1GHz | Processing speed are between 8MHz to 50MHz |
| Based on Von Neuman model, where program and data are stored in same memory model. | Based on Harvard architecture where program memory and data memory are separate. |

## 2. Explain ARM core data flow model with a neat diagram.



- 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.
  - Figure shows a Von Neumann implementation of the ARM—data items and instructions share the same bus. (In contrast, Harvard implementations of the ARM use two different buses).
- The instruction decoder translates instructions before they are executed. Each instruction executed belongs to a particular instruction set.
- The ARM processor, like all RISC processors, uses load-store architecture—means it has two instruction types for transferring data in and out of the processor:
  - load instructions copy data from memory to registers in the core.
  - store instructions copy data from registers to memory.
- There are no data processing instructions that directly manipulate data in memory. Thus, data processing is carried out in registers.
- Data items are placed in the register file—a storage bank made up of 32-bit registers.
  - Since the ARM core is a 32-bit processor, most instructions treat the registers as holding signed or unsigned 32-bit values. The sign extends hardware converts signed 8-bit and 16-bit numbers to 32-bit values as they are read from memory and placed in a register.
- ARM instructions typically have two source registers, Rn and Rm, and a single result or 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. Data processing instructions write the result in Rd directly to the register file.
- 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. Together the barrel shifter and ALU can calculate a wide range of expressions and addresses.
- 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 Incrementor updates the address register before the core reads or writes the next register value from or to the next sequential memory location.
- The processor continues executing instructions until an exception or interrupt changes the normal execution flow.

**3. Along with neat diagram of an ARM based embedded system (Microcontroller), explain the hardware components.**



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.

  - **Memory controller:** Memory controllers connect different types of memory to the processor bus. These memory devices allow the initialization code to be executed.

  - **Interrupt controller:** An interrupt controller provides a programmable governing policy that allows software to determine which peripheral or device can interrupt the processor at any specific time. There are two types of interrupt controller available for ARM processors. These are the standard interrupt controller and vector interrupt controller. The interrupt handler determines which device requires servicing by reading a device bitmap register in the interrupt controller.

- **Peripherals:** The peripherals provide all the input-output capability external to the chip and are responsible for the uniqueness of the embedded device. A peripheral device performs input and output functions for the chip by connecting to other devices or sensors that are off chip. All ARM peripherals are memory mapped. Controllers are specialized peripherals that implement higher levels of functionality within the embedded system.

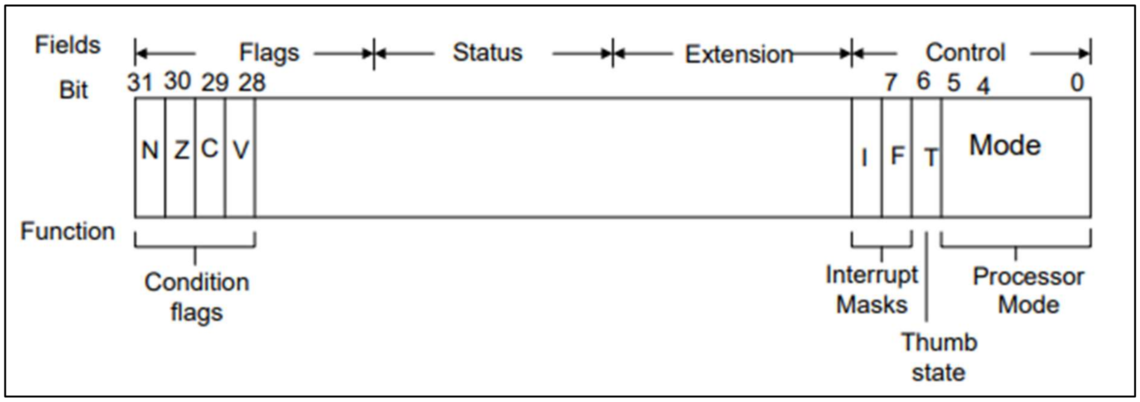
- **Bus:** A bus is used to communicate between different parts of the device. Embedded devices use an on-chip bus that is internal to the chip and that allows different peripheral devices to be interconnected with an ARM core. The ARM processor core is bus master-a logical device capable of initiating a data transfer with another device across the same bus.

- **Memory:** An embedded system has to have some form of memory to store and execute code. There are three types of memory- cache, main memory, and secondary memory. Cache is the fastest memory located near the ARM processor core. Cache is placed between the main memory and the core. It is used to speed up data transfer between the processor and main memory. The main memory is large depending on the application. Secondary memory is the largest and slowest form of memory.

**4. Explain the different processor modes provided by ARM7.**

- The processor mode determines which registers are active and the access rights to the CPSR register itself. Each processor mode is either privileged or nonprivileged.
- A privileged mode allows read/write access to the CPSR.
- A nonprivileged 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 nonprivileged mode.
- The privilege modes are abort, fast interrupt request , interrupt request, supervisor, system and undefined. The nonprivileged mode is user.
- The processor enters abort mode when there is a failed attempt to access memory.
- Fast interrupt request and interrupt request modes correspond to the two interrupt levels available on the ARM processor.
- Supervisor mode is the mode that the processor is in after reset and is generally the mode that an operating system kernel operates in.
- System mode is a special version of user mode that allows full read-write access to the CPSR.
- Undefined mode is used when the processor encounters an instruction that is undefined or not supported by the implementation.
- User mode is used for programs and applications.

**5. Give the schematic of a Current Program Status Register of ARM7 processor briefing the individual bits.**

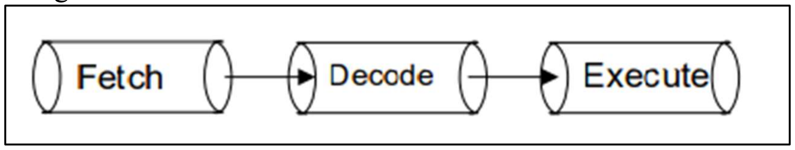


The CPSR is divided into four fields, each 8 bits wide: flags, status, extension, and control. In current designs the extension and status fields are reserved for future use. The control field contains the processor mode, state, and interrupts mask bits. The flag field contains the condition flags. The following table gives the bit patterns that represent each of the processor modes in the CPSR.
When CPSR bit 5 , T=1 , then the processor is in Thumb state. When T=0, the processor is in ARM state.
The CPSR has two internal mask bits,7 and 6 (I and F) which control the masking Interrupt request(IRQ) and Fast Interrupt Request(FIR).
The following table shows the conditional flags.

| **Flag** | **Flag Name** | **Set When** |
|---|---|---|
| N | Negative | Bit 31 of the result is a binary 1 |
| Z | Zero | The result is 0, frequently used to indicate equality |
| C | Carry | The result causes an unsigned carry |
| V | Overflow | The result causes a signed overflow |

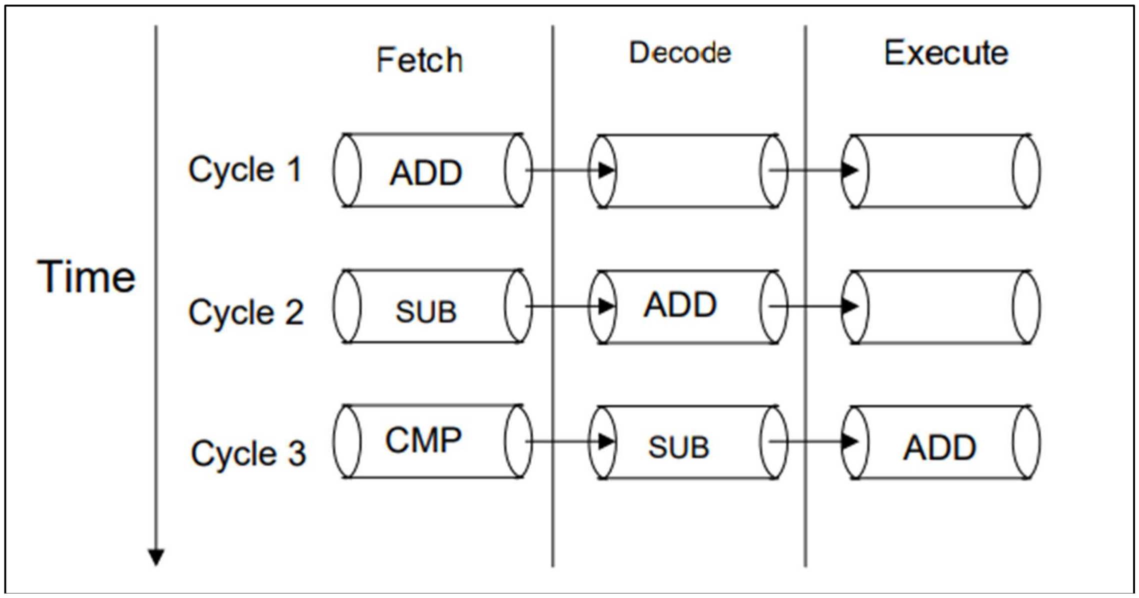
**6. What s Pipelining. Explain in detail schematically.**

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



- o Fetch loads an instruction from memory.
- o Decode identifies the instruction to be executed.
- o Execute processes the instruction and writes the result back to a register.

- Figure shown below illustrates the pipeline using a simple example. It shows a sequence of three instructions being fetched, decoded, and executed by the processor.
- Each instruction takes a single cycle to complete after the pipeline is filled.
- In the first cycle, the core fetches the ADD instruction from the memory.
- In the second cycle, the core fetches the SUB instruction and decodes the ADD instruction.
- In the third cycle, the core fetches CMP instruction from the memory, decodes the SUB instruction and executes the ADD instruction.



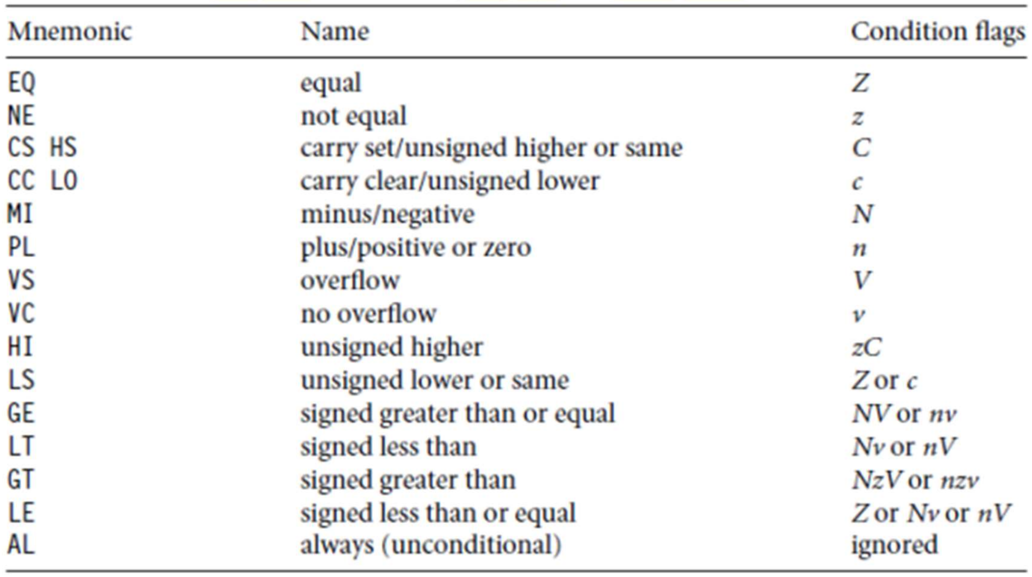
**7. Discuss the ARM design philosophy.**

Following are the ARM design philosophy:

- The ARM processor has been specially designed to be small to reduce power consumption and extend battery operation- essentially for applications such as mobile phones and personal digital assistants.
- High code density is a major requirement since embedded systems have limited memory due to cost and physical size restrictions. High code density is useful for applications that have limited on-board memory, such as mobile phones.
- Embedded systems are price sensitive and use low-cost memory devices. The ability to low-cost memory devices produces essential savings.
- Another requirement is to reduce the area of the die taken up by the embedded processor. For a single-chip solution, the smaller the area used by the embedded processor, the more available space for specialized peripherals.
- Another requirement is the hardware debug technology within the processor so that software engineers can view what is happening while the processor is executing code.

**8. Describe conditional execution. Write the different code suffix.**

- Conditional execution controls whether or not the core will execute an instruction.
- Prior to execution, the processor compares the condition attribute with the condition flags in the CPSR. If they match, then the instruction is executed; otherwise, the instruction is ignored.
- The condition attribute is post-fixed to the instruction mnemonic, which is encoded into the instruction.
- The following Table lists the conditional execution code mnemonics. When a condition mnemonic is not present, the default behavior is to set it to always (AL) execute.



| Mnemonic | Name | Condition flags |
|---|---|---|
| EQ | equal | *Z* |
| NE | not equal | *z* |
| CS HS | carry set/unsigned higher or same | *C* |
| CC LO | carry clear/unsigned lower | *c* |
| MI | minus/negative | *N* |
| PL | plus/positive or zero | *n* |
| VS | overflow | *V* |
| VC | no overflow | *v* |
| HI | unsigned higher | *zC* |
| LS | unsigned lower or same | *Z* or *c* |
| GE | signed greater than or equal | *NV* or *nv* |
| LT | signed less than | *Nv* or *nV* |
| GT | signed greater than | *NzV* or *nzv* |
| LE | signed less than or equal | *Z* or *Nv* or *nV* |
| AL | always (unconditional) | ignored |

**9. Differentiate between RISC and CISC processors.**

| CISC | RISC |
|---|---|
| Complex instructions, taking multiple clock | Simple instructions, taking single clock |
| Emphasis on hardware, complexity is in the micro-program/processor | Emphasis on software, complexity is in the complier. |
| Complex instructions, instructions executed by micro-program/processor | Reduced instructions, instructions executed by hardware |
| Variable format instructions, single register set and many instructions | Fixed format instructions, multiple register sets and few instructions |
| Many instructions and many addressing modes | Fixed instructions and few addressing modes |
| Conditional jump is usually based on status register bit | Conditional jump can be based on a bit anywhere in memory |
| Memory reference is embedded in many instructions | Memory reference is embedded in LOAD/STORE instructions |

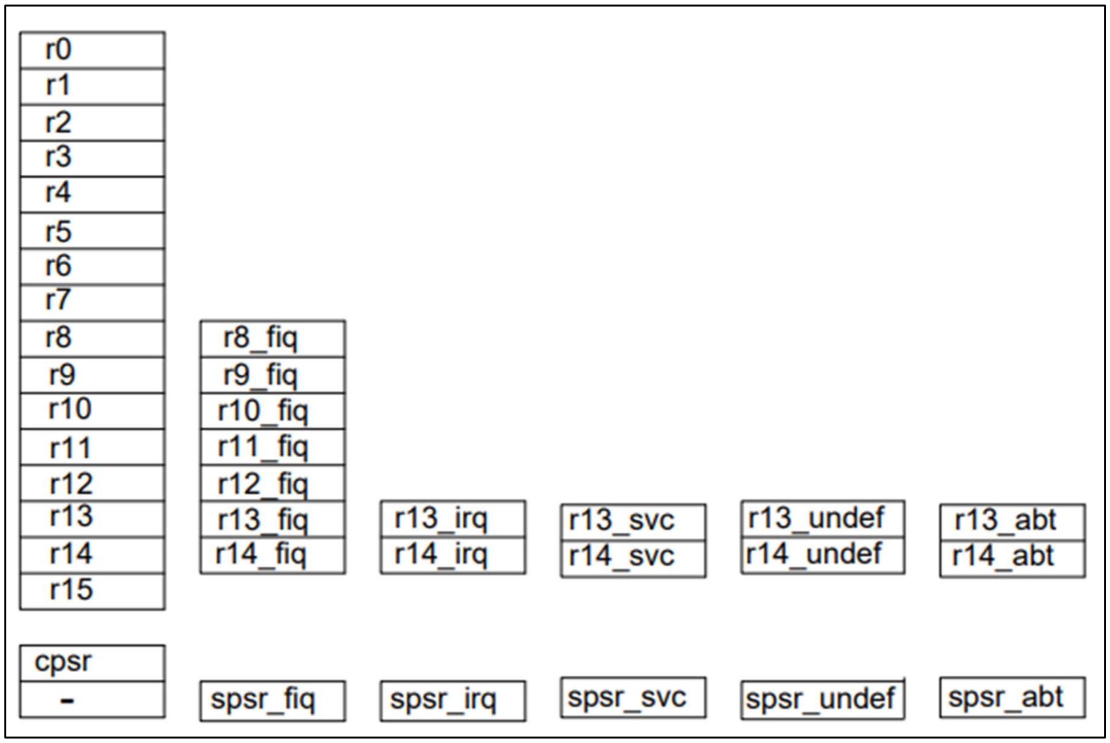
**10. Explain the major design rules to implement the RISC philosophy.**

- Instructions—RISC processors have a reduced number of instruction classes. These classes provide simple operations that can each be executed in a single cycle. The compiler or programmer synthesizes complicated operations (for example, a divide operation) by combining several simple instructions. Each instruction is having fixed length to allow the pipeline to fetch future instructions before decoding the current instruction.
  - In contrast, in CISC processors the instructions are often of variable size and take many cycles to execute.
- Pipelines—The processing of instructions is broken down into smaller units that can be executed in parallel by pipelines. Ideally the pipeline advances by one step on each cycle for maximum throughput. Instructions can be decoded in one pipeline stage.
  - There is no need for an instruction to be executed by a mini program called microcode as on CISC processors.
- Registers—RISC machines have a large general-purpose register set. Any register can contain either data or an address. Registers act as the fast local memory store for all data processing operations.
  - In contrast, CISC processors have dedicated registers for specific purposes.
- Load-store architecture—The processor operates on data held in registers. Separate load and store instructions transfer data between the register bank and external memory. Memory accesses are costly, so separating memory accesses from data processing provides an advantage because you can use data items held in the register bank multiple times without needing multiple memory accesses.
  - In contrast, with a CISC design the data processing operations can act on memory directly.
- These design rules allow a RISC processor to be simpler, and thus the core can operate at higher clock frequencies.
  - In contrast, traditional CISC processors are more complex and operate at lower clock frequencies.

**11. Briefly describe the concept of exceptions, interrupts, and the vector table.**
- When an exception or interrupt occurs, the processor sets the pc to a specific memory address. The address is within a special address range called the vector table.
  - The entries in the vector table are instructions that branch to specific routines designed to handle a particular exception or interrupt.
  - The memory map address 0x00000000 (or in some processors starting at the offset 0xffff0000) is reserved for the vector table, a set of 32-bit words.
- When an exception or interrupt occurs, the processor suspends normal execution and starts loading instructions from the exception vector table.
- Each vector table entry contains a form of branch instruction pointing to the start of a specific routine:
  - **Reset vector** is the location of the first instruction executed by the processor when power is applied. This instruction branches to the initialization code.
  - **Undefined instruction vector** is used when the processor cannot decode an instruction.
  - **Software interrupt vector** is called when you execute a SWI instruction. The SWI instruction is frequently used as the mechanism to invoke an operating system routine.
  - **Prefetch abort** vector occurs when the processor attempts to fetch an instruction from an address without the correct access permissions. The actual abort occurs in the decode stage.
  - **Data abort vector** is similar to a prefetch abort but is raised when an instruction attempts to access data memory without the correct access permissions.
  - **Interrupt request vector** is used by external hardware to interrupt the normal execution flow of the processor. It can only be raised if IRQs are not masked in the CPSR.
  - **Fast interrupt request vector** is similar to the interrupt request but is reserved for hardware requiring faster response times. It can only be raised if FIQs are not masked in the CPSR.

**12. Explain the programmer's model of ARM processors with complete register sets available.**

Figure below 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. 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.

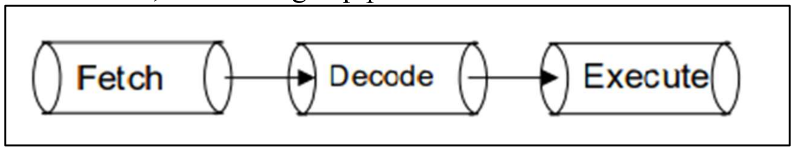


Every processor mode except user mode can change mode by writing directly to the mode bits of the CPSR. All processor modes except system mode have a set of associated banked registers that are subset of the main 16 registers. A banked register maps one-to-one onto a user mode register. If the processor mode is changed, a banked register from the new mode will replace an existing register. For example, when the processor mode is in the interrupt request mode, the instructions you execute still excess registers named r13 and r14. However, these registers are the banked registers r13_irq and r14_irq. The user mode registers r13_usr and r14_usr are not affected by the instruction referencing these registers. A program still has normal access to the other registers r0 to r12. The processor mode can be changed by a program that writes directly to the CPSR, when the processor core is in privilege mode. The following exception and interrupts cause a mode change: reset, interrupt request, fast interrupt request, software interrupt, data abort, prefetch abort and undefined instructions. Exceptions and interrupts suspend the normal execution of sequential instructions and jump to a specific location. The core changing from user mode to interrupt request mode, happens when an interrupt request occurs due to an external device raising an interrupt to the processor core. This change causes user registers r13 and r14 to be banked. The user registers are replaced with registers r13_irq and r14_irq respectively. r14_irq contains the return address and r13_irq contains the stack pointer for interrupt request mode.

**13. What is pipeline in ARM? Illustrate with an example. Show the pipeline stages of ARM7, ARM9 and ARM10.**

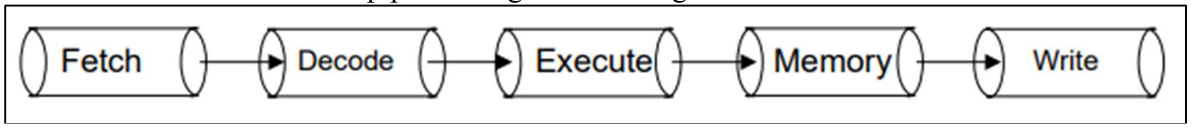
Pipeline is the mechanism to speed up execution by fetching the next instruction while other instructions are being decoded and executed.

In an ARM7, a three-staged pipeline is used.

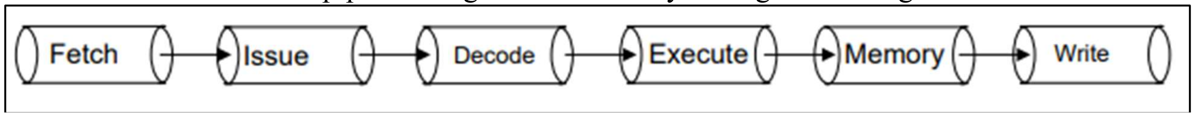


- Fetch loads an instruction from memory.
- Decode identifies the instruction to be executed.
- Execute processes the instruction and writes the result back to a register.

The ARM9 core increases the pipeline length to five stages.



The ARM10 increases the pipeline length still further by adding a sixth stage.



1. **Explain the MOV instruction set provided by ARM7 with the example for each.**

Move instruction 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
```

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

Example: This example shows a simple move instruction. The MOV instruction takes the contents of register r5 and copies them into register r7, in this case, taking the value 5, and overwriting the value 8 in register r7.

```
PRE:   r5 = 5
       r7 = 8
MOV r7, r5 ; let r7 = r5
POST:  r5 = 5
       r7 = 5
```

2. **Write a program for forward and backward branch by considering an example.**

```
        B forward
        ADD r1, r2, #4
        ADD r0, r6, #2
        ADD r3, r7, #4
forward
        SUB r1, r2, #4
--------------------------------------------------
backward
        ADD r1, r2, #4
        SUB r1, r2, #4
        ADD r4, r6, r7
        B backward
```

3. **Write and explain arithmetic instructions with respect to the ARM processor.**

The arithmetic instructions implement addition and subtraction of 32-bit signed and unsigned values.

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

Example: The following simple subtract instruction subtracts a value stored in register r2 from a value stored in register r1. The result is stored in register r0.

```
PRE     r0 = 0x00000000
        r1 = 0x00000002
        r2 = 0x00000001
SUB r0, r1, r2;
POST    r0 = 0x00000001
```

Example: The SUBS instruction is useful for decrementing loop counters. In this example, we subtract the immediate value one from the value one stored in register r1. The result value zero is written to register r1. The CPSR is updated with the ZC flags being set.

```
PRE     cpsr = nzcvqiFt_USER
        r1 = 0x00000001
SUBS r1, r1, #1
POST    cpsr = nZCvqiFt_USER
        r1 = 0x00000000
```

4. **Design ARM assembly language program to perform the addition and multiplication of two 32-bit numbers.**

**Multiplication:**

```
        area multi, code, readonly
entry
start
        ldr r0, = 11111111
        ldr r1, = 2222222
        mul r2,r1,r0
stop b stop
        end
```

**Addition:**

```
        area add, code, readonly
entry
start
        ldr r0, =11111111
        ldr r1, =22222222
        add r2,r1,r0
stop b stop
        end
```

5. **Explain the different branch instructions of ARM processor.**

A branch instruction changes the flow of execution or is used to call a routine. This type of instruction allows programs to have subroutines, if-then-else structures, and loops. The change of execution flow forces the program counter pc to point to a new address.

- o The address label is stored in the instruction as a signed pc-relative offset and must be within approximately 32 MB of the branch instruction.
- o T refers to the Thumb bit in the CPSR. When instructions set T, the ARM switches to Thumb state.

Example: This example shows a forward and backward branch. Because these loops are address specific, we do not include the pre- and post-conditions. The forward branch skips three instructions. The backward branch creates an infinite loop.

```
        B forward
        ADD r1, r2, #4
        ADD r0, r6, #2
        ADD r3, r7, #4
forward
        SUB r1, r2, #4
-------------------------------------------------
backward
        ADD r1, r2, #4
        SUB r1, r2, #4
        ADD r4, r6, r7
        B backward
```

- In this example, forward and backward are the labels. The branch labels are placed at the beginning of the line and are used to mark an address that can be used later by the assembler to calculate the branch offset.
- The branch with link, or BL, instruction is similar to the B instruction but overwrites the link register lr with a return address. It performs a subroutine call.

**6. Explain the different barrel shifter operations with suitable examples.**

- Data processing instructions are processed within the arithmetic logic unit (ALU).
- A unique and powerful 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.
- Pre-processing or shift occurs within the cycle time of the instruction.
    - This shift increases the power and flexibility of many data processing operations.
    - This is particularly useful for loading constants into a register and achieving fast multiplies or divisions by a power of 2.

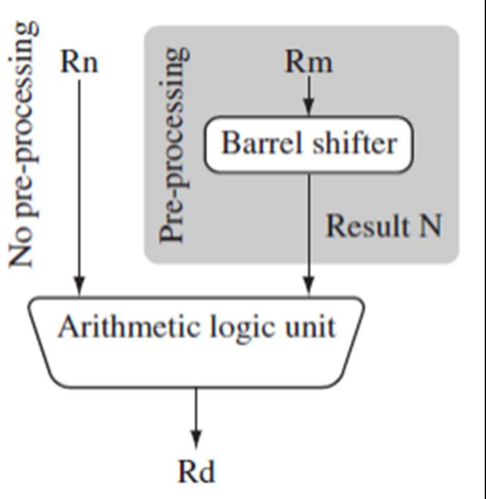


**Figure: Barrel Shifter and ALU**

- Figure shows the data flow between the ALU and the barrel shifter.
- Register Rn enters the ALU without any pre- processing of registers.
- We apply a logical shift left (LSL) to register Rm before moving it to the destination register. This is the same as applying the standard C language shift operator « to the register.
- The MOV instruction copies the shift operator result N into register Rd. N represents the result of the LSL operation described in the following Table.