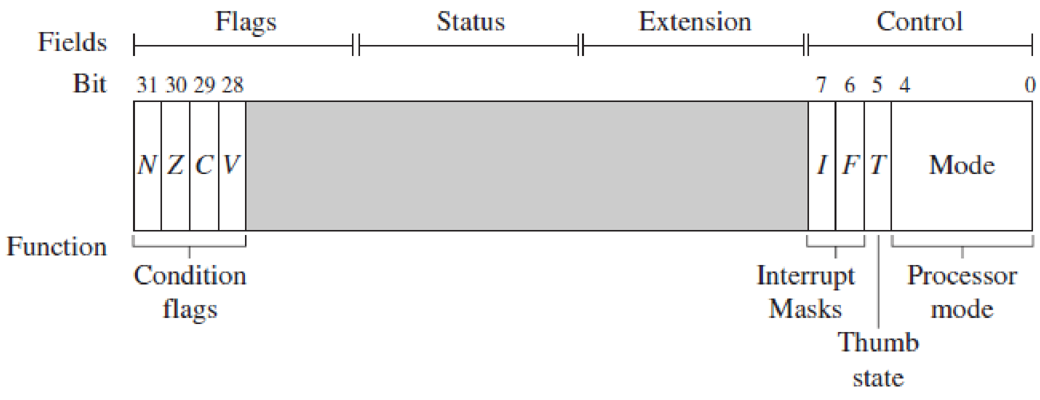
11) CURRENT PROGRAM STATUS REGISTER:

The ARM core uses the *cpsr* to monitor and control internal operations. The *cpsr* is a dedicated 32-bit register and resides in the register file. The following Figure shows the basic layout of a generic program status register. Note that the shaded parts are reserved for future expansion.

A Generic Program Status Register (psr)



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 *interrupt mask* bits.
- The ***flags field*** contains the *condition flags*.

Some ARM processor cores have extra bits allocated. For example, the *J bit*, which can be found in the flags field, is only available on *Jazelle-enabled processors*, which execute 8-bit instructions. It is highly probable that future designs will assign extra bits for the monitoring and control of new features.

Processor Modes:

The processor mode determines which registers are active and the access rights to the *cpsr* register itself. Each processor mode is either privileged or non-privileged:

- A *privileged mode* allows full read-write access to the *cpsr*.
- A *non-privileged mode* only allows read access to the control field in the *cpsr*, but still allows read-write access to the condition flags.

There are *seven processor modes* in total:

six privileged modes (abort, fast interrupt request, interrupt request, supervisor, system, and undefined)

•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.

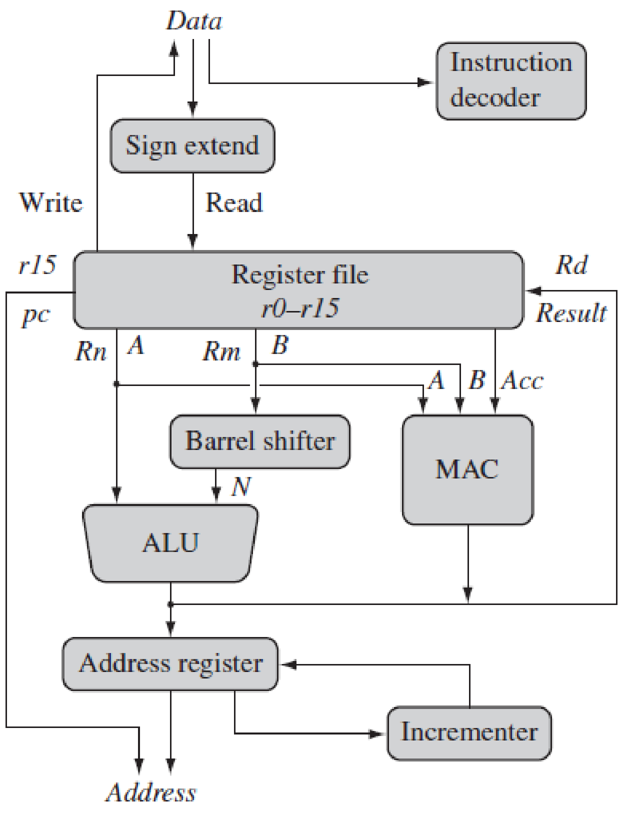
•***System mode*** is a special version of user mode that allows full read-write access to the *cpsr*.

•***Undefined mode*** is used when the processor encounters an instruction that is undefined or not supported by the implementation.

one non-privileged mode (user).

•***User mode*** is used for programs and applications.

12)



ARM Core dataflow Model

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
- 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 extend hardware converts signed 8-bit and16-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

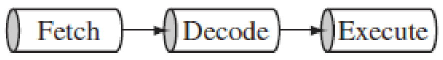
- r*destination register*, *Rd*. Source operands are read from the register file using the internal buses A and B, respectively.
- The *ALU (arithmetic logic unit)* or *MAC (multiply-accumulate unit)* takes the register values *Rn*and *Rm* from the A and B buses and computes a result. Data processing 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 pre processed 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 *Incrementer* 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.

13) **PIPELINE:**

- A *pipeline* is the mechanism in a RISC processor, which is used to execute instructions.
- Pipeline speeds up execution by fetching the next instruction while other instructions are being decoded and executed.



ARM7 Three-stage Pipeline The above Figure shows a 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.

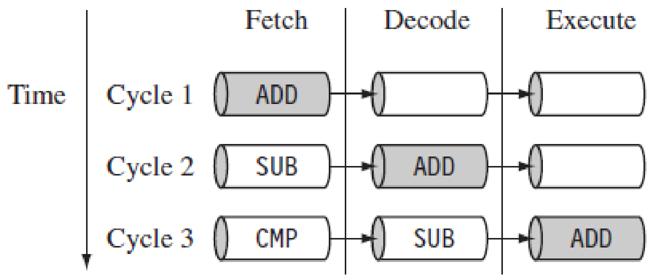


Figure: Pipelined Instruction Sequence

The Figure shows a sequence of three instructions being fetched, decoded, and executed by the processor.

The three instructions are placed into the pipeline sequentially.

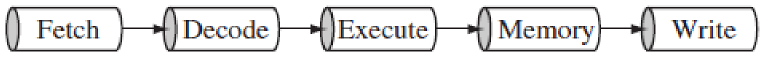
- In the first cycle, the core fetches the ADD instruction from memory.
- In the second cycle, the core fetches the SUB instruction and decodes the ADD instruction.
- In the third cycle, both the SUB and ADD instructions are moved along the pipeline. The ADD instruction is executed, the SUB instruction is decoded, and the CMP instruction is fetched.

This procedure is called ***filling the pipeline***.

The pipeline allows the core to execute an instruction every cycle.

- As the pipeline length increases, the amount of work done at each stage is reduced, which allows the processor to attain a higher operating frequency. This in turn *increases the performance*.
- The increased pipeline length also means increased *system latency* and there can be *data dependency* between certain stages.

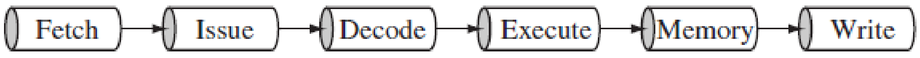
- The pipeline design for each ARM family differs. For example, The ARM9 core increases the pipeline length to five stages, as shown in Figure



ARM9 Five-stage Pipeline

The ARM9 adds a memory and writeback stage, which allows the ARM9 to –process on average 1.1 Dhrystone MIPS per MHz increase the instruction throughput in ARM9 by around 13% compared with anARM7.

The ARM10 increases the pipeline length still further by adding a sixth stage, as shown in the following Figure.



ARM10 Six-stage Pipeline

The ARM10 –can process on average 1.3 Dhrystone MIPS per MHz have about 34% more throughput than an ARM7 processor core but again at a higher latency cost.