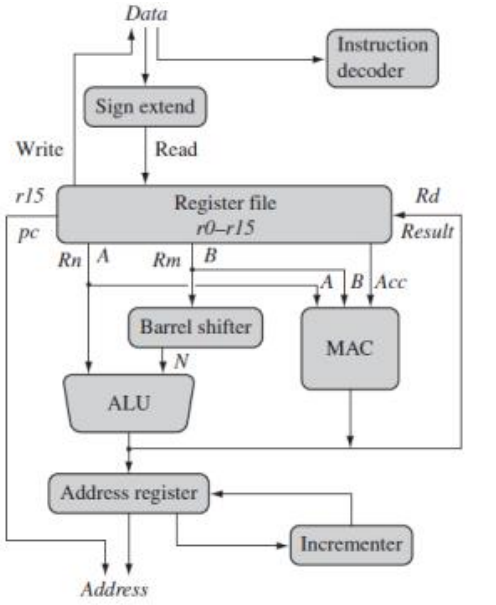
1a.

Data flow

- A programmer can think of an ARM core as functional units connected by data buses.
- The arrows represent the flow of data,
- The lines represent the buses,
- The boxes represent either an operation unit or a storage area.
- The figure shows not only the flow of data but also the abstract components that make up an ARM core.



ARM core dataflow model.

- Data enters the processor core through the *Data* bus. The data may be an instruction to execute or a data item.
- 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.
- The ARM processor, like all RISC processors, uses a *load-store architecture*. This 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, and conversely the store instructions copy data from registers to memory.
- 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 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) & 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.
- 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.

1b.

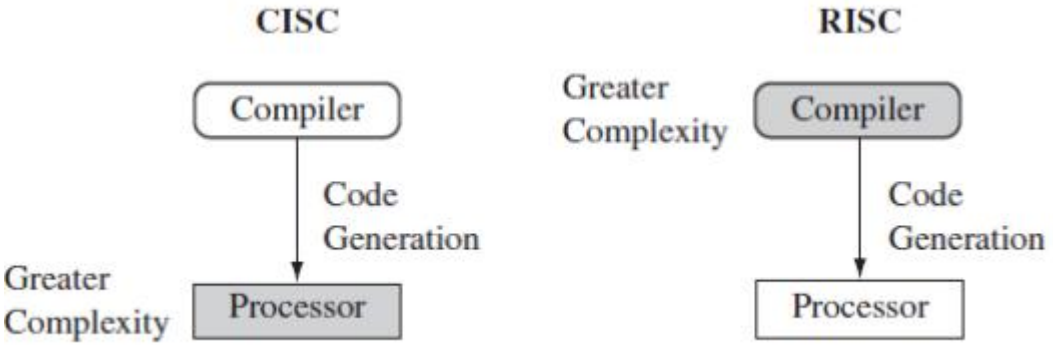
- RISC design philosophy is aimed at delivering simple but powerful instructions that execute within a single cycle at a high clock speed.
- Main idea behind is to make hardware simpler by using an instruction set composed of a few basic steps for loading, evaluating, and storing operations, intelligence is provided in software rather than hardware.
- RISC philosophy is implemented with 4 major design rules,

i. Instructions: RISC class provide simple operations that can each execute in a single cycle. The compiler or programmer synthesizes complicated operations of fixed length allowing pipelining.

ii. 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.

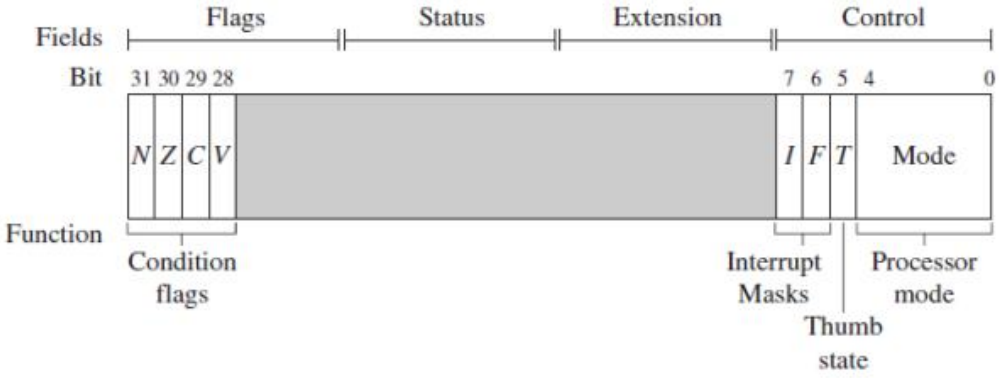
iii. Registers: RISC machines have a large general-purpose register set. Any register can contain either data or an address. Registers act as the fast local memory store

iv. 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.



1c.

CPSR

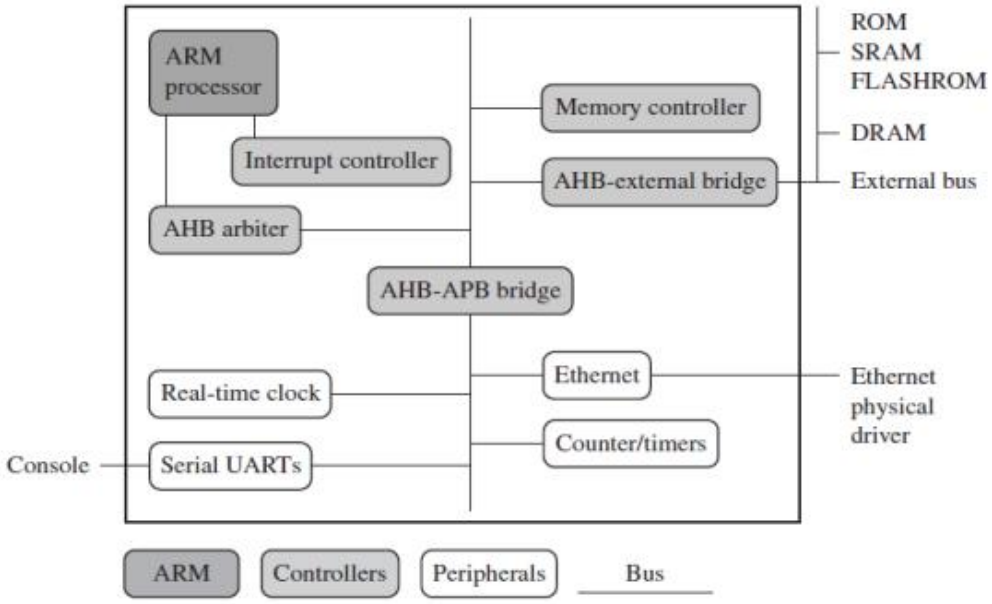


A generic program status register (*psr*).

- ARM core uses the cpsr to monitor and control internal operations.
- The cpsr is divided into four fields, each 8 bits wide: flags, status, extension, and control.
- The control field contains the processor mode, state, and interrupt mask bits.
- The flags field contains the condition flags.

2a.

ARM based embedded device



An example of an ARM-based embedded device, a microcontroller.

- Device can be divided into four main hardware components,

i. The *ARM processor* controls the embedded device. Different versions of the ARM processor are available to suit the desired operating characteristics. An ARM processor comprises a core (the execution engine that processes instructions and manipulates data) plus the surrounding components that interface it with a bus. These components can include memory management and caches.

ii. *Controllers* coordinate important functional blocks of the system. Two commonly found controllers are interrupt and memory controllers.

iii. The *peripherals* provide all the input-output capability external to the chip and are responsible for the uniqueness of the embedded device.

iv. A *bus* is used to communicate between different parts of the device.

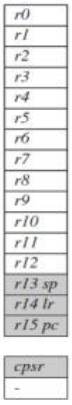
2b.

- Variable cycle execution for certain instructions—Not every ARM instruction executes in a single cycle.
  Ex: load-store-multiple instructions vary in the number of execution cycles depending upon the number of registers being transferred.
  Code density is also improved since multiple register transfers are common operations at the start and end of functions.
- Inline barrel shifter leading to more complex instructions—The inline barrel shifter is a hardware component that preprocesses one of the input registers before it is used by an instruction. This expands the capability of many instructions to improve core performance and code density.
- Thumb 16-bit instruction set—ARM enhanced the processor core by adding a second 16-bit instruction set called Thumb that permits the ARM core to execute either 16- or 32-bit instructions. The 16-bit instructions improve code density by about 30% over 32-bit fixed-length instructions.
- Conditional execution—An instruction is only executed when a specific condition has been satisfied. This feature improves performance and code density by reducing branch instructions.
- Enhanced instructions—The enhanced digital signal processor (DSP) instructions were added to the standard ARM instruction set to support fast 16×16-bit multiplier operations and saturation. These instructions allow a faster-performing ARM processor in some cases to replace the traditional combinations of a processor plus a DSP.

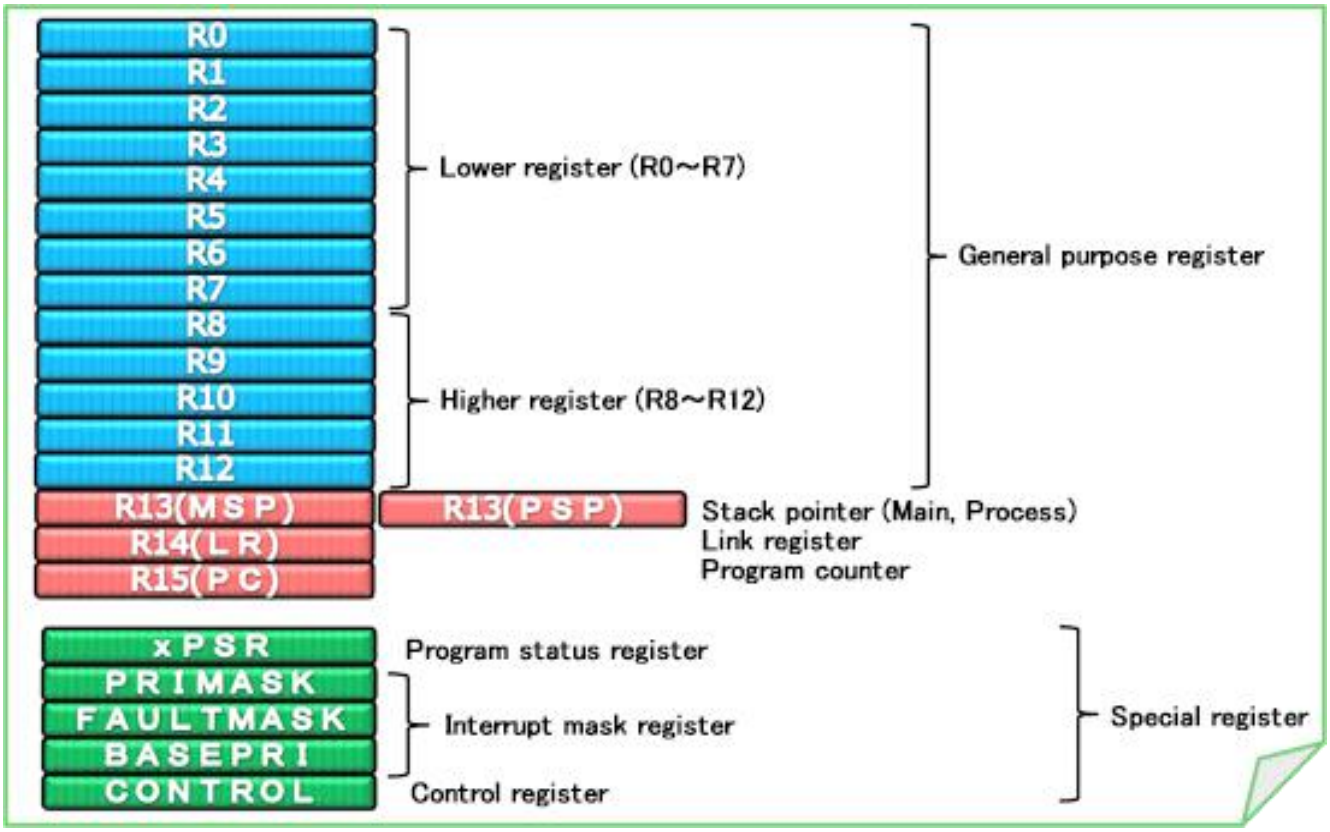
2c.

Registers

- General-purpose registers hold either data or an address. They are identified with the letter r prefixed to the register number.
- Ex: register 4 is given the label r4.
- There are up to 18 active registers:
  - 16 data registers and 2 processor status registers.
  - data registers are visible to the programmer as r0 to r15.
- Register r13 is traditionally used as the stack pointer (sp) and stores the head of the stack in the current processor mode.
- Register r14 is called the link register (lr) and is where the core puts the return address whenever it calls a subroutine.
- Register r15 is the program counter (pc) and contains the address of the next instruction to be fetched by the processor.



Registers available in *user* mode.



3a.

a.MOVr0,r1 & MOVSr0,r1,LSL#1

b.TESTr0,r3 & ANDr0,r1,r1,LSL#1

c.CMP r1,r3 & SUB r0,r1,r2

Explanation +example

3b.

Processor modes

1. The processor enters *abort* mode when there is a failed 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 generally 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 programs and applications.

Mode	Abbreviation	Privileged	Mode[4:0]
Abort	abt	yes	10111
Fast interrupt request	fiq	yes	10001
Interrupt request	irq	yes	10010
Supervisor	svc	yes	10011
System	sys	yes	11111
Undefined	und	yes	11011
User	usr	no	10000

3c.

AMBA Bus Protocol

- Acronym for Advanced Microcontroller Bus Architecture (AMBA).
- Introduced in 1996.
- Widely adopted as the on-chip bus architecture used for ARM processors.
- The first AMBA buses introduced were
  I. ARM System Bus (ASB)
  II. ARM Peripheral Bus (APB)
  III. Later ARM introduced another bus design, called the ARM High Performance Bus (AHB).
- Peripheral designers can reuse the same design on multiple projects. Because there are a large number of peripherals developed with an AMBA interface, hardware designers have a wide choice of tested and proven peripherals for use in a device.
- plug-and-play interface for hardware developers improves availability and time to market.
- AHB provides higher data throughput than ASB because it is based on a centralized multiplexed bus scheme rather than the ASB bidirectional bus design. This change allows the AHB bus to run at higher clock speeds.
- ARM has introduced two variations on the AHB bus:
  I. Multi-layer AHB
  II. AHB-Lite.

4a.

a.BIC

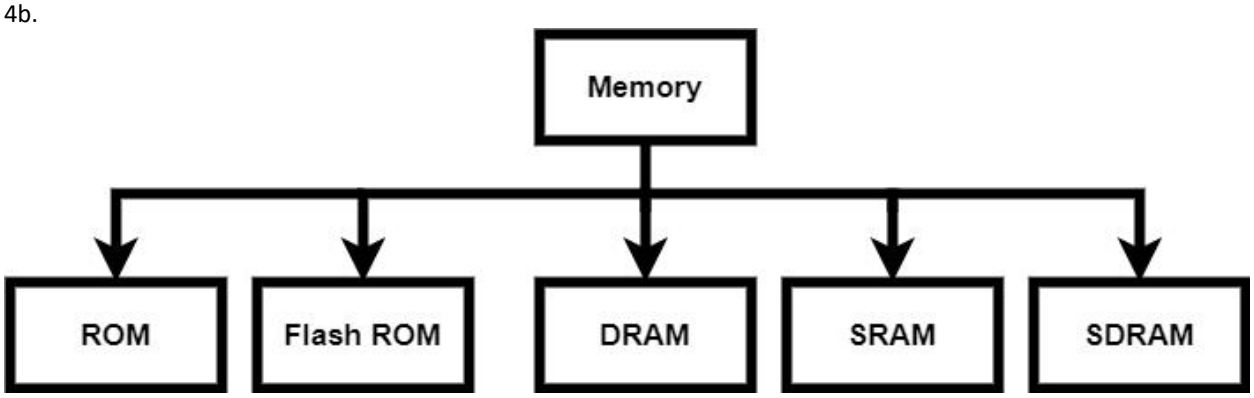
b.RSB

c.MVN

d.UMULL

Explanation + example

4b.



Explanation of every block

4c.

Core extensions

- The hardware extensions covered in this section are standard components placed next to the ARM core.
- They improve performance, manage resources, and provide extra functionality and are designed to provide flexibility in handling particular applications.
- Each ARM family has different extensions available.
- There are three hardware extensions ARM wraps around the core:
  1. cache and tightly coupled memory,
  2. memory management, and
  3. the coprocessor interface.