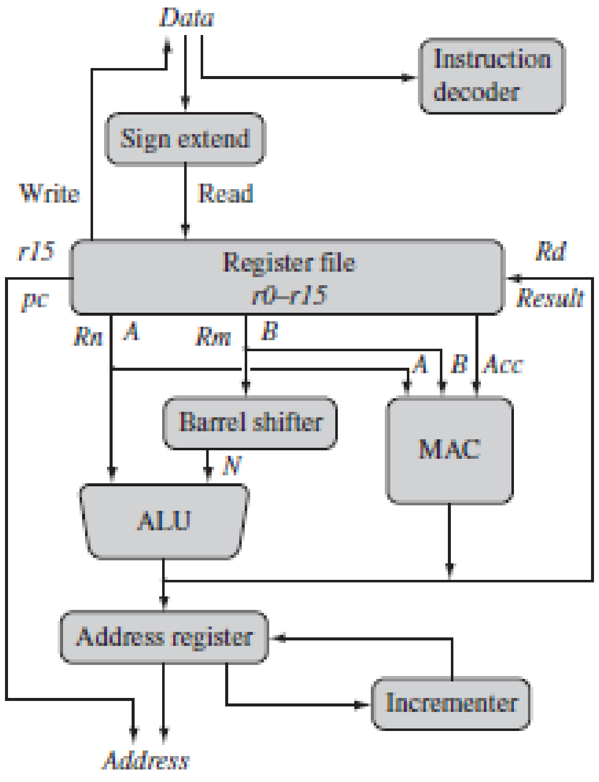
ARM Processor Fundamentals

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.
There are no data processing instructions that directly manipulate data in memory. Thus, data processing is carried out solely 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 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.



ARM core dataflow model.

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 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.
Now that you have an overview of the processor core we'll take a more detailed look at some of the key components of the processor: the registers, the current program status register (*cpsr*), and the pipeline.

Registers

General-purpose registers hold either data or an address. They are identified with the letter ***r*** prefixed to the register number. For example, register 4 is given the label *r4.*

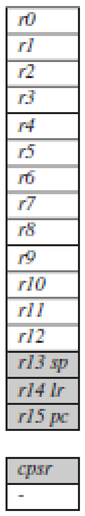


Figure 2.2 Registers available in *user* mode.

Figure 2.2 shows the active registers available in *user* mode—a protected mode normally used when executing applications. The processor can operate in seven different modes, which we will introduce shortly. All the registers shown are 32 bits in size.

There are up to 18 active registers: 16 data registers and 2 processor status registers. The data registers are visible to the programmer as *r0* to *r15.*

The ARM processor has three registers assigned to a particular task or special function: *r13*, *r14*, and *r15*. They are frequently given different labels to differentiate them from the other registers.

In Figure 2.2, the shaded registers identify the assigned special-purpose registers:

■ 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.

Depending upon the context, registers *r13* and *r14* can also be used as general-purpose registers, which can be particularly useful since these registers are banked during a processor mode change. However, it is dangerous to use *r13* as a general register when the processor is running any form of operating system because operating systems often assume that *r13* always points to a valid stack frame.

In addition to the 16 data registers, there are two program status registers: *cpsr* and *spsr* (the current and saved program status registers, respectively). The register file contains all the registers available to a programmer. Which registers are visible to the programmer depend upon the current mode of the processor.

Current Program Status Register - cpsr

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 *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.

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 nonprivileged:
A privileged mode allows full read-write access to the *cpsr*. Conversely, a nonprivileged 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*) and one **nonprivileged** (USER) mode

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.