Central Processing Unit

Processor Organization

To understand the organization of the processor, we need to understand the tasks that a processor must do.

Fetch instruction: The processor reads an instruction from memory (register, cache, main memory).

Interpret instruction: The instruction is decoded to determine what action is required.

Fetch data: The execution of an instruction may require reading data from memory or an I/O module.

Process data: The execution of an instruction may require performing some arithmetic or logical operation on data.

Write data: The results of an execution may require writing data to memory or an I/O module.
To do these things, it should be clear that the processor needs to store some data temporarily. In other words, the processor needs a small internal memory.

The figure alongside is a simplified view of a processor, indicating its connection to the rest of the system via the system bus. The major components of the processor are an arithmetic and logic unit (ALU) and a control unit (CU). The ALU does the actual computation or processing of data. The control unit controls the movement of data and instructions into and out of the processor and controls the operation of the ALU.
In addition, the figure shows a minimal internal memory, consisting of a set of storage locations, called registers. The data transfer and logic control paths are indicated, including internal processor bus which is needed to transfer data between the various registers and the ALU because the ALU in fact operates only on data in the internal processor memory.

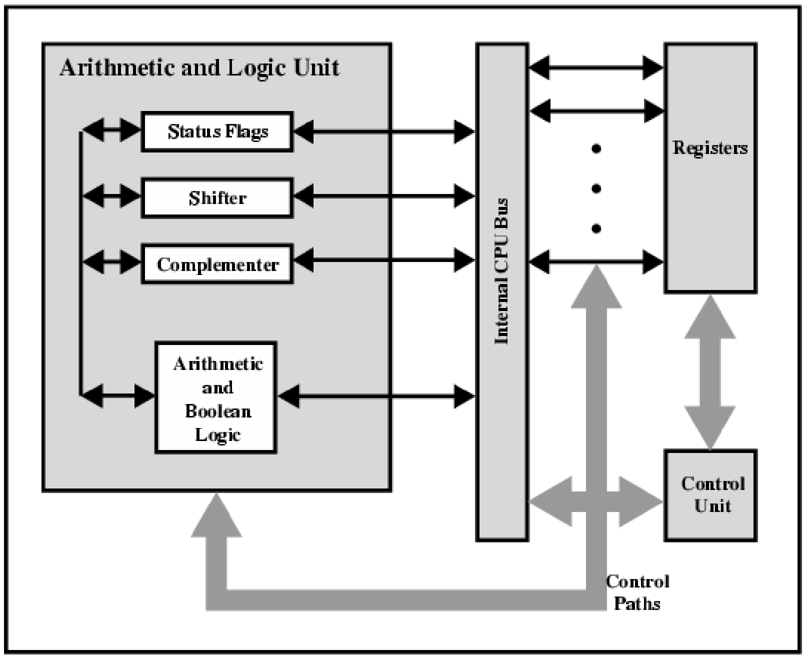


Fig. Internal Structure of CPU

Register Organization

Register organization is the arrangement of the registers in the processor. The processor designers decide the organization of the registers in a processor. Different processors may have different register organization. Depending on the roles played by the registers they can be categorized into two types, user-visible register and control and status register.

Before learning register organization in brief, let us discuss what is register?

What is Register?

Registers are the *smaller* and the *fastest* accessible memory units in the central processing unit (CPU). According to memory hierarchy, the registers in the processor, function a level above the main memory and cache memory. The registers used by the central unit are also called as processor registers.

A register can hold the *instruction, address location, or operands*. Sometimes, the instruction has register as a part of itself.

Types of Registers

As we have discussed above, registers can be organized into two main categories i.e. the User-Visible Registers and the Control and Status Registers. Although we can't separate the registers in the processors clearly among these two categories.

This is because in some processors, a register may be user-visible and in some, the same may not be user-visible. But for our rest of discussion regarding register organization, we will consider these two categories of register.

1. **User Visible Registers**
   - General Purpose Register
   - Data Register
   - Address Register
   - Condition Codes

2. **Control and Status Registers**
   - Program Counter
   - Instruction Register
   - Memory Address Register
   - Memory Buffer Register

User-Visible Registers

These registers are visible to the assembly or machine language programmers and they use them effectively to minimize the memory references in the instructions. Well, these registers can only be referenced using the machine or assembly language.

The registers that fall in this category are discussed below:

1. General Purpose Register

The general-purpose registers detain either the addresses or the data. Although we have separate data registers and address registers. The general purpose register also accepts the intermediate results in the course of program execution.

Well, the programmers can restrict some of the general-purpose registers to specific functions. Like, some registers are specifically used for stack operations or for floating-point operations. The general-purpose register can also be employed for the addressing functions**.**

2. Data Register

The term itself describes that these registers are employed to hold the data. But the programmers can't use these registers for calculating operand address.

3. Address Register

Now, the address registers contain the address of an operand or it can also act as a general-purpose register. An address register may be dedicated to a certain addressing mode. Let us understand this with the examples.

(a) Segment Pointer Register
A memory divided in segments, requires a segment register to hold the base address of the segment. There can be multiple segment registers. As one segment register can be employed to hold the base address of the segment occupied by the operating system. The other segment register can hold the base address of the segment allotted to the processor.

(b) Index Register
The index register is employed for indexed addressing and it is initial value is 0. Generally, it used for traversing the memory locations. After each reference, the index register is incremented or decremented by 1, depending upon the nature of the operation.
Sometime the index register may be auto indexed.

(c) Stack Pointer Register
The stack register has the address that points the stack top.

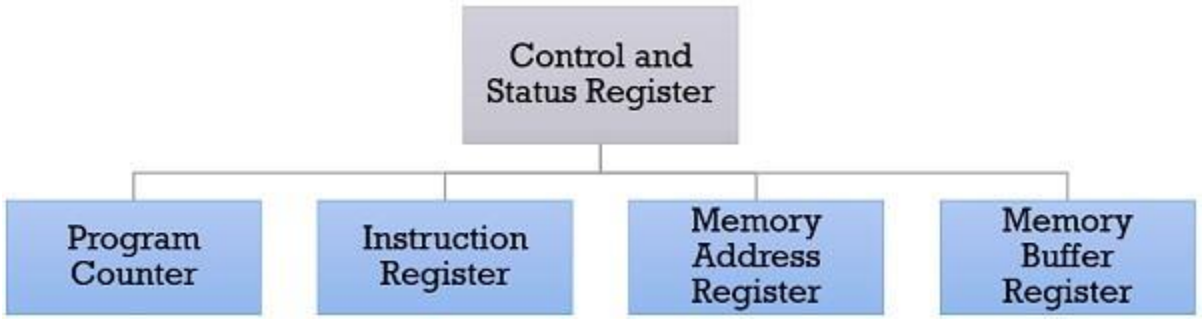
***4.* Condition Code**

Condition codes are the flag bits which are the part of the control register. The condition codes are set by the processor as a result of an operation and they are implicitly read through the machine instruction.

The programmers are not allowed to alter the conditional codes. Generally, the condition codes are tested during conditional branch operation.

Control and Status Registers

The control and status register holds the address or data that is important to control the processor's operation. The most important thing is that these registers are not visible to the users. Below we will discuss all the control and status registers are essential for the execution of an instruction**.**



1. Program Counter

The program counter is a processor register that holds the address of the instruction that has to be executed next. It is a processor which updates the program counter with the address of the next instruction to be fetched for execution.

2. Instruction Register

Instruction register has the instruction that is currently fetched. It helps in analyzing the opcode and operand present in the instruction.

3. Memory Address Register (MAR)

Memory address register holds the address of a memory location.

4. Memory Buffer Register (MBR)

The memory buffer register holds the data that has to be written to a memory location or it holds the data that is recently been read.

The memory address registers (MAR) and memory buffer registers (MBR) are used to move the data between processor and memory.

Apart from the above registers, several processors have a register termed as Program Status Word (PSW). As the word suggests it contains the status information.

The fields included in **Program Status Word (PSW)**:

- **Sign:** This field has the *resultant sign bit* of the last arithmetic operation performed.
- **Zero:** This field is set when the result of the operation is *zero*.
- **Carry:** This field is set when an arithmetic operation results in a *carry into* or *borrow out*.
- **Equal:** If a *logical operation* results in, *equality* the Equal bit is set.
- **Overflow:** This bit indicates the *arithmetic overflow*.
- **Interrupt:** This bit is set to *enable* or *disable* the interrupts.
- **Supervisor:** This bit indicates whether the processor is executing in the *supervisor mode* or the *user mode*.

So, these are the types of registers a processor has. The processor designer organizes the registers according to the requirement of the processor.

Arithmetic and Logic Unit

An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer. Modern CPUs contain very powerful and complex ALUs. In addition to ALUs, modern CPUs contain a control unit (CU).

Whenever calculations are required, the control unit transfers the data from storage unit to ALU. When the operations are done, the result is transferred back to the storage unit.

Most of the operations of a CPU are performed by one or more ALUs, which load data from input registers. A register is a small amount of storage available as part of a CPU. The control unit tells the ALU what operation to perform on that data, and the ALU stores the result in an output register. The control unit moves the data between these registers, the ALU, and memory.

An ALU performs basic arithmetic and logic operations. Examples of arithmetic operations are addition, subtraction, multiplication, and division. Examples of logic operations are comparisons of values such as NOT, AND, and OR.

DATA PATHS

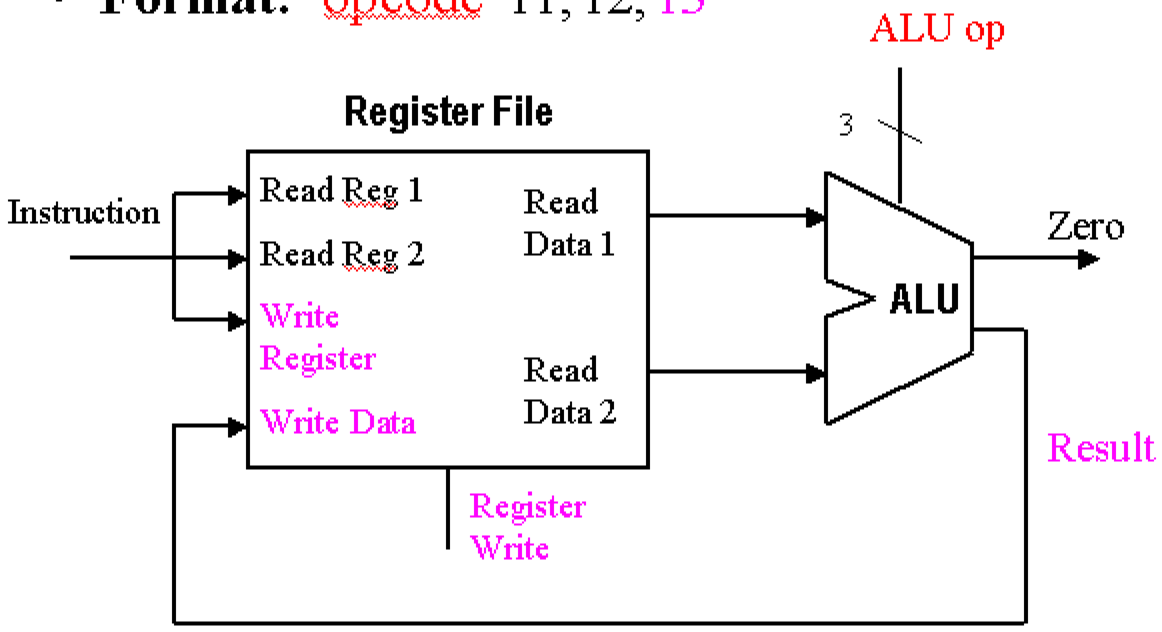
A data path (also written as datapath) is a set of functional units that carry out data processing operations. Data paths make up a Central Processing Unit (CPU) along with a control unit. In other words, Datapath is the hardware that performs all the required operations, for example, ALU, registers, and internal buses.

Various Computer Architectures imply various Data Paths for their instructions. We will be discussing the data paths for various instruction formats used in MIPS Architecture.

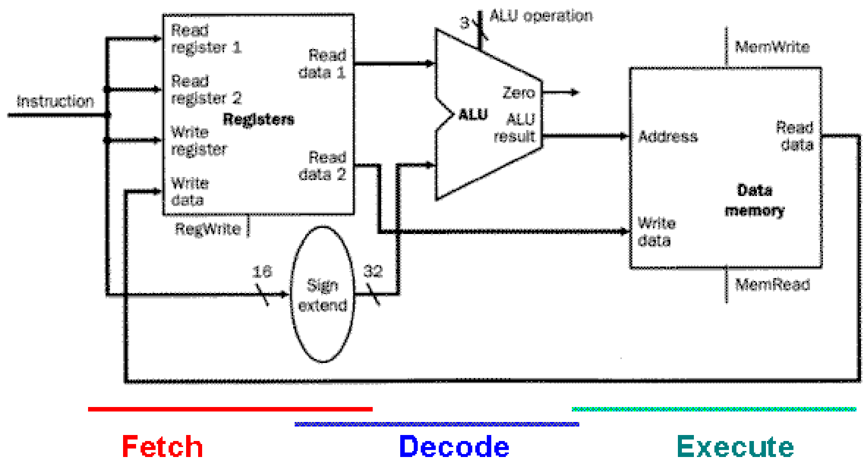
Datapath for R- Type Instructions

Implementation of the datapath for R-format instructions is fairly straightforward - the register file and the ALU are all that is required. The ALU accepts its input from the DataRead ports of the register file, and the register file is written to by the ALUresult output of the ALU, in combination with the RegWrite signal.

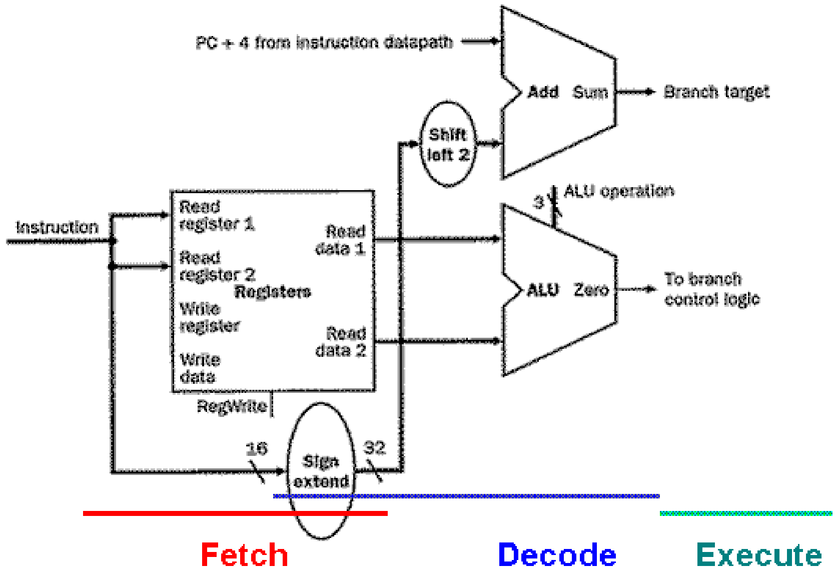
- **Format:** opcode r1, r2, r3



Datapath for I- Type Instructions (load/store)
The lw and sw each compute an address by adding an offset to the value in a register. Thus, they need to reference the register file and the ALU to perform the addition. Note the value of one register is added to the sign extended offset value in the instruction. On a store the data from the other register is sent to memory. On a load the data read from memory is placed in the register file at the location specified in the second register.
The load/store datapath takes operand #1 (the base address) from the register file, and sign-extends the offset, which is obtained from the instruction input to the register file. The sign-extended offset and the base address are combined by the ALU to yield the memory address, which is input to the address port of the data memory. The MemRead signal is then activated, and the output data obtained from the ReadData port of the data memory is then written back to the Register File using its WriteData port, with RegWrite asserted.



Datapath for J- Type Instructions (Branching Instructions)
The branch datapath takes operand #1 (the offset) from the instruction input to the register file, then sign-extends the offset. The sign-extended offset and the program counter (incremented by 4 bytes to reference the next instruction after the branch instruction) are combined by ALU #1 to yield the branch target address. The operands for the branch condition to evaluate are concurrently obtained from the register file via the ReadData ports, and are input to ALU #2, which outputs a one or zero value to the branch control logic.



For Example:
The branch datapath (jump is an unconditional branch) uses instructions such as beq $t1, $t2, offset, where offset is a 16-bit offset for computing the branch target address via PC-relative addressing. The beq instruction reads from registers $t1 and $t2, then compares the data obtained from these registers to see if they are equal. If equal, the branch is taken. Otherwise, the branch is not taken.

(For Instruction cycle check chapter 5)