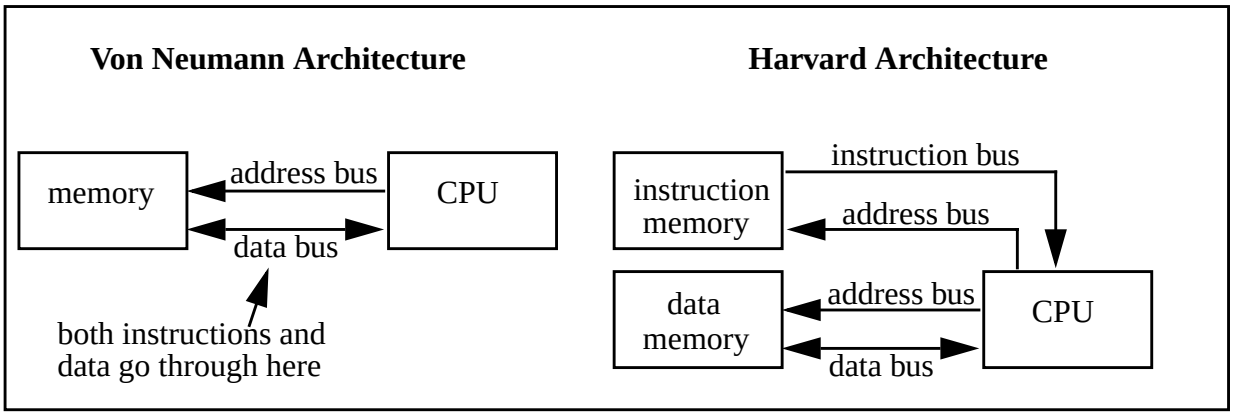
A Brief view of Computer Architecture

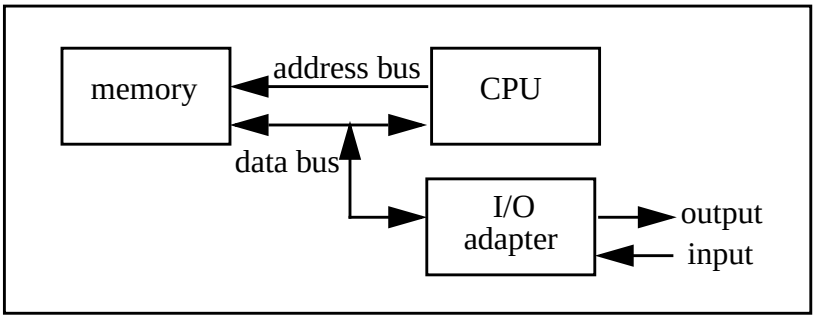
We do not intend to give a full explaination of computer architecture. However, we will develop some intuitive models of how computers work. Let';s start at a high level picture of how a computer could be structured and work down into the inner workings of the CPU from there.

The simplest representation of a computer, one with only a CPU (central processing unit) and a memory unit is shown below. This computer can't do very much as it cannot effect its environment outside of its own memory.



In the figure above we see two different architectural styles. The Von Neumann architecture is representative of most computer architectures. The most distinguishing feature of the Von Neumann computer is that it stores both data and instruction information in the same memory. The Harvard architecture computer stores instructions and data in separate memory. Many microcontroller architectures (like ones you would find in a toaster) are Harvard architectures.

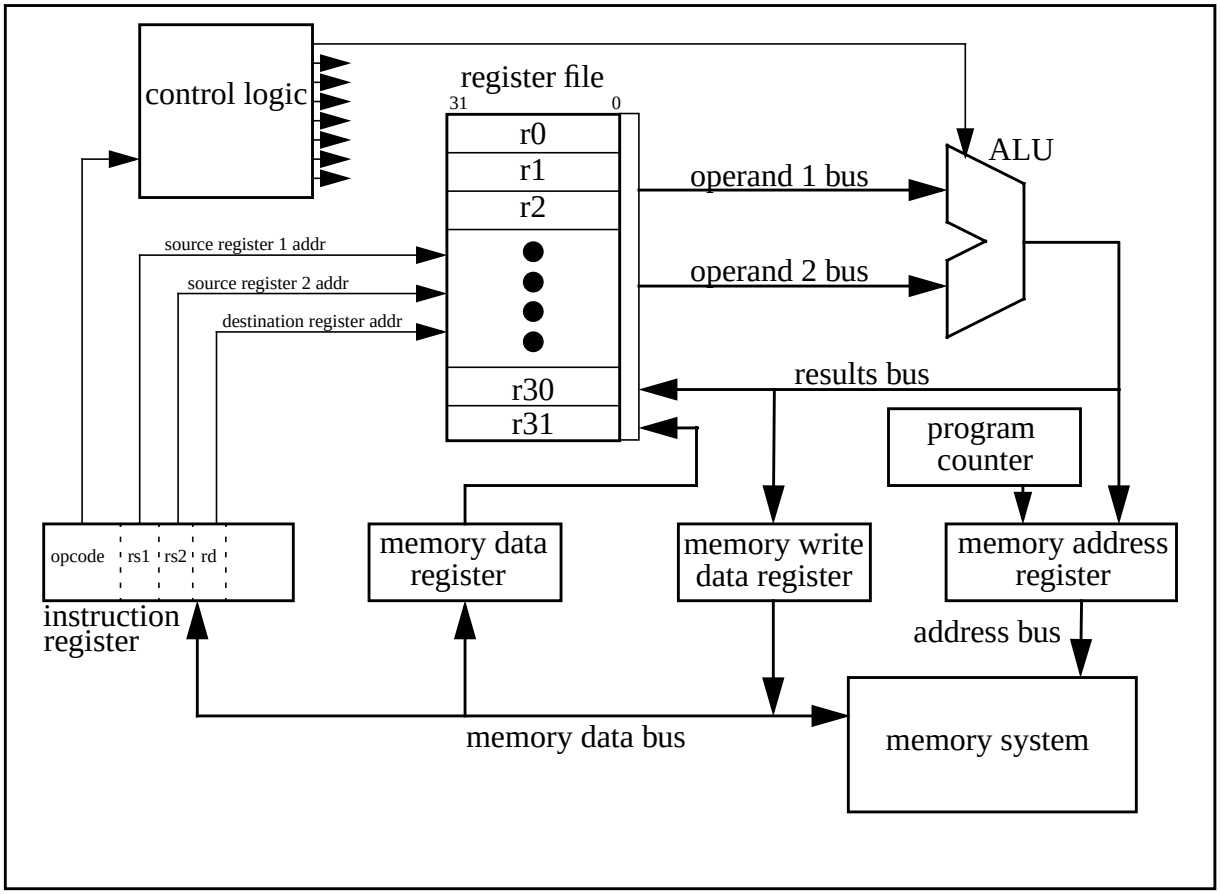
As just mentioned, a computer system shown above could not turn on a light or pop your toast out before it burns. It needs an I/O (input/output) port. Let's add one to a Von Neumann architecture.



Now we have a resonable computer system. It can read instructions from a memory, execute them, store the results and control external devices. The I/O adapter in the most simple terms forms a temporary storage area where outputs may be posted and inputs may be held until the CPU can check them.

Note that we are not talking about PC architecture. A PC's architecture is one example of a computer architecture, it is not computer architecture in general.

Lets explore the CPU of a Von Neumann machine. The CPU is the brain of the computer where lots of interesting things are happening.



The CPU executes a program that is fetched more or less sequentially from the memory system. To execute each instruction a number of steps are taken.

1. Fetch the instruction from the location from memory system specified by the memory address register into the instruction register.
2. Decode the instruction, and increment the PC.
3. Fetch the operands.
4. Execute the desired operation using the ALU
5. Optionally access memory
6. Store the result in the desired location (could be a register or memory)

The program counter (PC) is a special register that holds the address of the next instruction to be fetched. To fetch an instruction, the contents of the PC is placed on the memory address bus and a read of memory is performed. When the instruction returns, it is placed in the instruction register.

Once the instruction is written into the instruction register, it is decoded to determine what the instruction is and what action is to be taken. During this time, the PC is incremented to the

next instructions address in preparation for fetching it.

The decoding of the instruction will determine the major category of the instruction. For this example, (which is like many RISC microprocessors) there are three categories of instruction. These are *memory access*, *calculation* and *branching*.

Memory access instructions are only for bringing data from memory to the register file or for putting data into memory from a register. Loading a register from memory is called a *load* operation. Storing data to memory is called a *store* operation.

Calculation instructions use one or more operands in registers or in the instruction itself and generate a result using the ALU. These instructions are performed entirely within the CPU and do not access memory. Typically the calculation instructions are operations like add or subtract, or boolean operations like OR and AND.

Branch instructions are the decision making instructions. They have the ability to alter the flow of a program depending on the results of a comparison. They also use the ALU. For example, a *branch not equal* instruction might compare the values to two operands held in the register file and if they are not equal (as determined by a subtract operation), the PC will be loaded with a new address other than the *next* one in the instruction sequence.

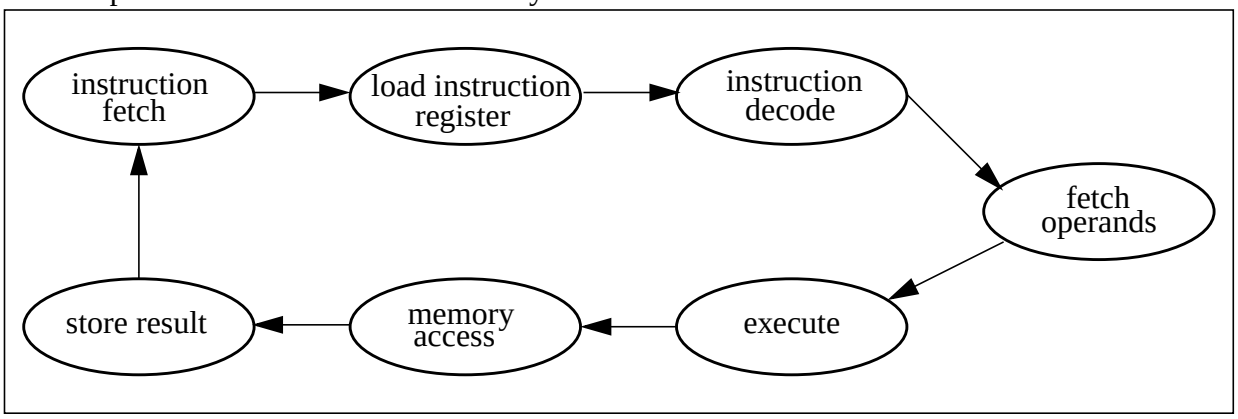
Once the decoding of the instruction is done, the operands must be accessed. This step is usually done in parallel with the decoding. The operands are accessed by the addresses for them imbedded within the instruction. The operands are held in the register file at a certain location. This location is accessed by fields within the instruction. In our example, three fields of five bits each will allow simultaneous access to three of the 32 register locations. Once the operand registers are accessed, they data from them flows directly to the ALU for processing.

The execution step is usually a simple combinatorial manipulation of the two operands. The action that the ALU takes is dictated by the opcode field in the instruction. Determining what action the ALU takes on the operands is part of the decoding process. If the instruction is a branch, the ALU serves two purposes, one to do some comparison, the other to calculate the branch address. For memory access instructions, the ALU is used to compute the memory address of the operand to be loaded or stored.

Following the execution step is the memory access step used only by the memory access instructions. If a load or store is performed, the operand address is put on the memory address bus and the read or write signals to the memory are asserted. After some delay the data is written or read from memory.

The last step in the execution of an instructions will only be taken by the store instruction. Once the operand is read from memory, we store it into the memory data register. From here, in the last step the operand is stored into the register file.

This process of execution can be easily viewed as a state machine model.



RISC versus CISC

There is another way to distinguish computer architectures. The competing architectural models are known as RISC (reduced instruction set computer) and (complex instruction set computer). The major differences are summarized below.

Feature	RISC	CISC
Instruction size	1 word	1 to 54 bytes
Execution time	1 clock	1 to 100s of cycles
Addressing modes	small	large
Work done per instruction	small	varies
Instruction count/program	large	smaller

During the nineteen eighties, a great controversy existed over the advantages and disadvantages of each style of architecture. In the end, RISC seems to have to won. Even the notable surviving CISC architecture, Intel's x86 family, has adopted many ideas from the RISC camp to maintain its performance.