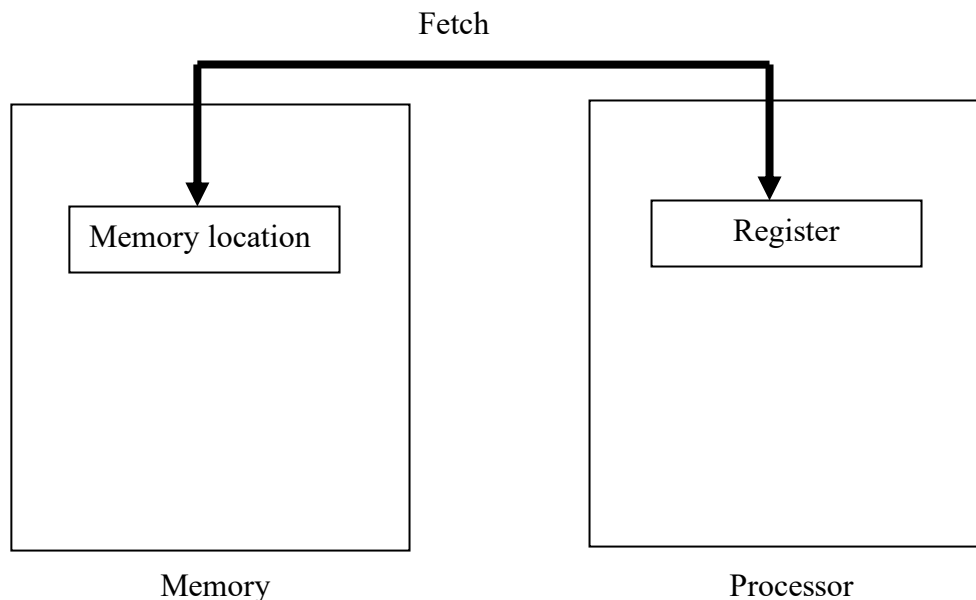


Processing Stored Data

Processors handle data one piece at a time. Each item has to be obtained (fetched), placed in a location where it can be used (stored in a register), acted upon, then placed somewhere else (saved) so that the scarce resource, the register, can be used again.



The fundamental question of how fast an item of data can be processed is pretty obviously made up of at least eleven different elements:

- (i) The time taken to identify where it is;
- (ii) The time taken to obtain control of the communication channel;
- (iii) The time taken to put the data on the communication channel;
- (iv) The time taken to transfer it;
- (v) The time taken to store it in the register;
-
- (vi) The time taken to apply the operation to the data in the register;
-
- (vii) The time taken to identify where to put it;
- (viii) The time taken to obtain control of the communication channel;
- (ix) The time taken to put it back on the communication channel;
- (x) The time taken to transfer it back to the memory device;
- (xi) The time taken to store it in the location

The first and last five activities might appear symmetrical. They are not, unless the memory and the processor are symmetrical in their speed of operation.

In the world of physical things, we could minimise the time taken to get things to and from the processor by putting things in a pile and always taking the next item, and storing things in a pile always placing things on top; or we could offload the task of organising things in sequence for use or storage to another device. We could dedicate the communication channel just to transferring between memory and the processor. We could make the channel as fast as

possible, and place the memory as close as possible to the processor, to minimise the transfer time. We can minimise the time taken to store the datum in the register or memory by making the device as fast as possible.

If we are being really speedy, we can have sufficient registers not to need any external memory (but will still need some means of getting data into and out of the registers in the first place, otherwise we have created an electronic sulking machine).

To go further, we can use banks of processors, to be able to do more than one item of data concurrently.

Ultimately, we can get sensible and make sure everything we process needs processing, and is processed in the way that takes least time to achieve the result we desire.

In effect, this is the whole problem laid bare. Everything we talk about features some aspect of overcoming or minimising the impact of these difficulties, at different scales in terms of volumes of data, speeds of transfer and complexities of access. The mismatches between domains of time are significant. Take, for example, a human as an input device, where an expert typist might just get up to 200 words per minute. If each character is a datum, average word length 6 letters, we might be able to type in 1400 characters (remember the spaces are also characters) per minute, or 23 characters per second. The processor in the keyboard is probably capable of handling 23 million characters per second. If it is a simple USB keyboard its transfer protocol to the storage mechanism is capable of around 2 million characters per second.

So if we're looking for fast processing, we need to be thoughtful as to how we build up our systems. Transfer is only ever at the speed of the slowest link in the communication chain. If we're looking for speed of performance, it is controlled as much about what we process as it is about how we process it. As we push the boundaries of performance, devices and transfer mechanisms reach their physical limits; selection of what to process then becomes the defining parameter.

In essence then speed is about managing data. The processing elements, storage elements and the data channels make up the media through which we work. As we shall see later, this is in some ways simplistic, in that it focuses on the part of the activity which is physical. However, whatever we do in design of information processing, any improvement in hardware performance will always deliver speed increases for any given combination of programme and data. It may well be that we can speed things up more by careful consideration of what needs processing, by eliminating unnecessary data, by design of the sequence of work at the information level, rather than optimising the hardware processing.

Von Neumann's Stored Process Concept and its Consequence: the Fetch, Decode, Execute Cycle

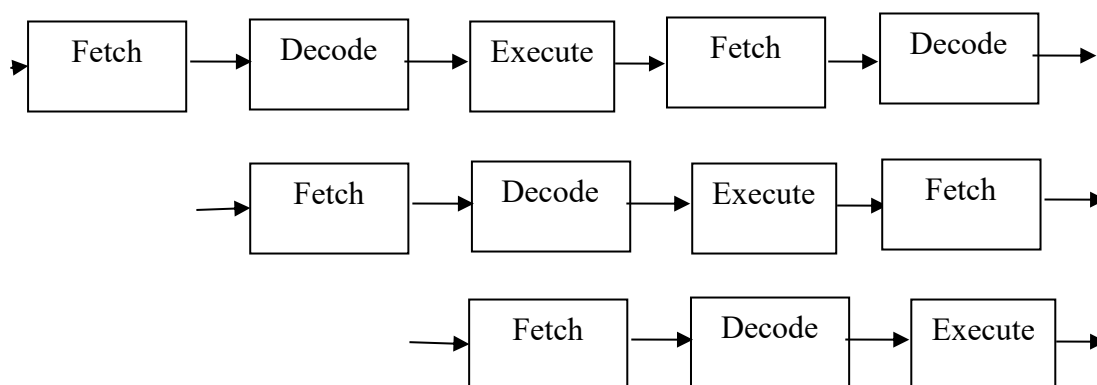
Von Neumann's ideas form the basics of all computer systems. The basic format can be summarised as:

- A main memory which stores both the data to be operated upon and the operating instructions
- An arithmetical and logical unit (ALU) capable of operating on binary data
- A control unit which interprets the instructions in memory and causes them to be executed
- Input and output capabilities controlled by the control unit

The fetch, decode, execute cycle follows from this arrangement. In order to operate, an instruction must be fetched from memory into the control unit, decoded into the required operations, then those operations executed. If done sequentially:



While decoding and executing, the fetch circuitry is not in use; similarly decode and execute are not operating while instruction fetching is underway. One obvious way of speeding this up is to run the operations in parallel, as illustrated below. This is called “pipelining”. Given the objective is to execute as nearly continuously as possible, having prefetched, predecoded instructions, i.e. a prefetch queue, is clearly beneficial.



Many programs contain loops and discontinuities. At this point, the prefetch queue will need to be flushed (emptied). Efficient pipelining therefore requires forward analysis of the code to determine likely changes and cater for them where possible.