3.3.6 Parallel processing

Parallel processing is the simultaneous use of more than one CPU to execute a program. Ideally, parallel processing makes a program run faster because there are more CPUs running it. In practice, it is often difficult to divide a program in such a way that separate CPUs can execute different portions without interfering with each other.

Most computers have just one CPU, but computers can have upto 1000 CPU's. Note that parallel processing differs from **multitasking**, in which a single CPU executes several programs at once.

There are a number of ways to carry out parallel processing; the table below shows each one of them and how they are applied in real life.

Types of parallel processing	Class of computer	Application
Pipeline	Single Instruction Single Data (SISD)	Inside a CPU
Array Processor	Single Instruction Multiple Data SIMD	Graphics cards, games consoles
Multi-Core	Multiple Instruction Multiple Data MIMD	Super computers, modern multi-core chips
Pipeline	Multiple Instruction Single Data MISD	Rarely Implemented

Advantages of parallel processing over the Von Neumann architecture

- Faster when handling large amounts of data, with each data set requiring the same processing (array and multi-core methods)
- Is not limited by the bus transfer rate (the Von Neumann bottleneck)
- Can make maximum use of the CPU (pipeline method) in spite of the bottleneck

Disadvantages

- Only certain types of data are suitable for parallel processing. Data that relies on the result of a previous operation cannot be made parallel. For parallel processing, each data set must be independent of each other.
- More costly in terms of hardware - multiple processing blocks needed, this applies to all four methods

3.3.6 Parallel processing

Pipelining:

Using the Von Neumann architecture for a microprocessor illustrates that basically an instruction can be in one of three phases. It could be being **fetched** (from memory), **decoded** (by the control unit) or being **executed** (by the control unit).

An alternative is to split the processor up into three parts, each of which handles one of the three stages. This would result in the situation shown in Fig. 3.3.d.1, which shows how this process, known as pipelining, works. Where each single line is a pipeline.

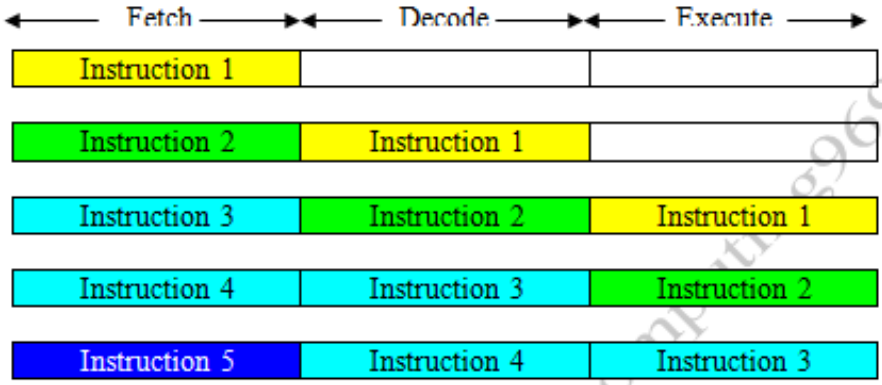
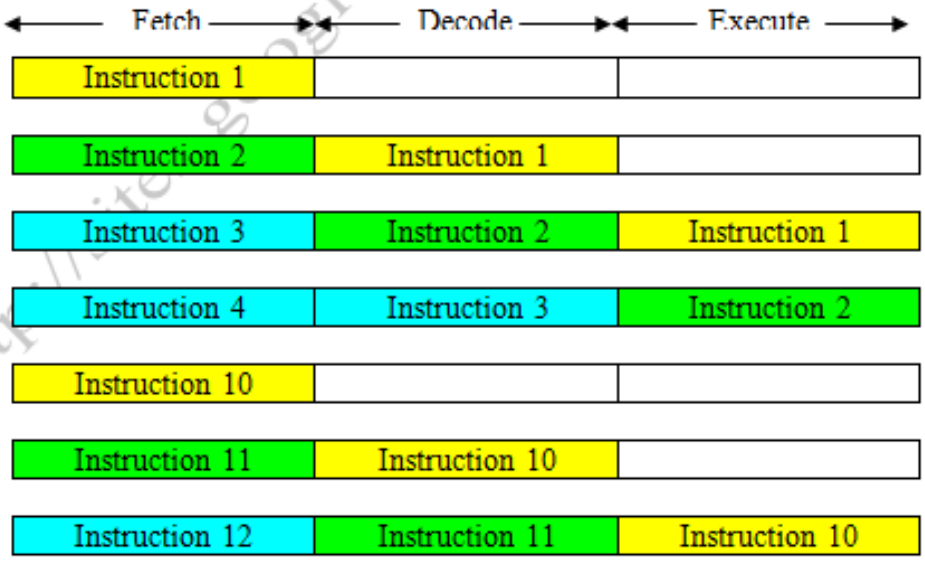


Fig. 3.3.d.1

This helps with the speed of throughput unless the next instruction in the pipe is not the next one that is needed. Suppose Instruction 2 is a jump to Instruction 10. Then Instructions 3, 4 and 5 need to be removed from the pipe and Instruction 10 needs to be loaded into the fetch part of the pipe. Thus, the pipe will have to be cleared and the cycle restarted in this case. The result is shown in Fig. 3.3.d.2 below

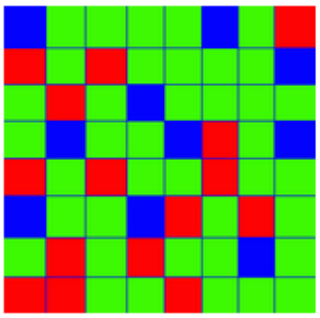


3.3.6 Parallel processing

The most apparent advantage of pipelining is that there are three instructions being dealt with at the same time. This SHOULD reduce the execution times considerably (to approximately 1/3 of the standard times), however, this would only be true for a very linear program. Once jump instructions are introduced the problem arises that the wrong instructions are in the pipeline waiting to be executed, so every time the sequence of instructions changes, the pipe line has to be cleared and the process started again.

SISD computers have one processor that handles one algorithm using one source of data at a time. The computer tackles and processes each task in order, and so sometimes people use the word “sequential” to describe SISD computers. They aren’t capable of performing parallel processing on their own.

Array or Vector processing:

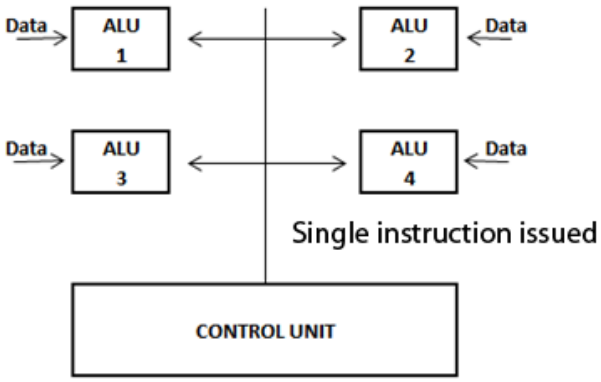


Some types of data can be processed independently of one another. A good example of this is the simple processing of pixels on a screen. If you wanted to make each colored pixel a different color according to what it currently holds. For example "Make all Red Pixels Blue", "Make Blue Pixels Red", "Leave Green pixels alone".

Some types of data can be processed independently of one another. A good example of this is the simple processing of pixels on a screen. If you wanted to make each colored pixel a different color according to what it currently holds. For example "Make all Red Pixels Blue", "Make Blue Pixels Red", "Leave Green pixels alone".

An array processor (or vector processor) has a number of Arithmetic Logic Units (ALU) that allows all the elements of an array to be processed at the same time.

The illustration Fig. 3.3.d.3 below shows the architecture of an array or vector processor



An array processor - a Single Instruction Multiple Data computer

3.3.6 Parallel processing

With an array processor, a single instruction is issued by a control unit and that instruction is applied to a number of data sets at the same time.

An array processor is a Single Instruction Multiple Data computer or SIMD

You will find games consoles and graphics cards making heavy use of array processors to shift those pixels about.

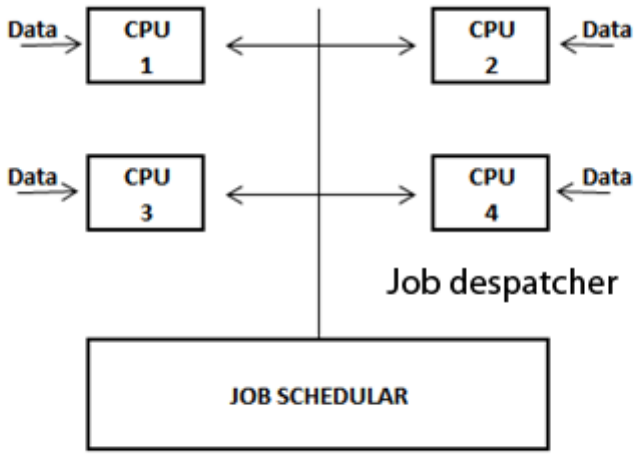
SIMD computer have several processors that follow the same set of instructions. SIMD computers run different data through the same algorithm. This can be useful for analyzing large chunks of data based on the same criteria.

Limitations of Array Processing

This architecture relies on the fact that the data sets are all acting on a single instruction. However, if these data sets somehow rely on each other then you cannot apply parallel processing. For example, if data A has to be processed before data B then you cannot do A and B simultaneously. This dependency is what makes parallel processing difficult to implement. And it is why sequential machines are still extremely common.

Multiple Processors:

Moving on from an array processor, where a single instruction acts upon multiple data sets and the next level of parallel processing is to have multiple instructions acting upon multiple data sets. This is achieved by having a number of CPUs being applied to a single problem, with each CPU carrying out only part of the overall problem.



Multi-core computer - Multi-Instructions Multiple Data computer

3.3.6 Parallel processing

A good example of this architecture is a supercomputer. For example the massively parallel IBM Blue Gene supercomputer that has 4,098 processors, allowing for 560 Teraflops of processing. This is applied to problems such as predicting climate change or running new drug simulations. Large problems that can be broken down into smaller sub-problems.

But even the humble CPU chip in your personal computers is likely to have multiple cores. For example the Intel Core 2 Duo has two CPUs (called 'cores') inside the chip, whilst the Quad core has four. A multi-core computer is a 'Multiple Instruction Multiple Data' computer or MIMD.

A **MIMD** has multiple processors, each capable of accepting its own instruction stream independently from the others. Each processor also pulls data from a separate data stream. An MIMD computer can execute several different processes at once. MIMD are more flexible than SIMD computers, but it is more difficult to make these systems work.

Limitations of multi-core processing

This architecture is dependent on being able to cut a problem down into chunks, each chunk can then be processed independently. But not many problems can be broken down in this way and so it remains a less used architecture.

Furthermore, the software programmer has to write the code to take advantage of the multi-core CPUs. This is actually quite difficult and even now most applications running on a multi-core CPU such as the Intel 2 Duo will not be making use of both cores most of the time.

A MISD computer has multiple processors. Each processor uses a different algorithm but uses the same shared input data. MISD computers can analyze the same set of data using several different operations at the same time. The number of operations depends upon the number of processors. There aren't many examples of MISD computers, partly because the problems a MISD computer can calculate are uncommon and specialized.

3.3.6 Parallel processing

Maths co-processor

So far, we have discussed parallel processing as a means of speeding up data processing. This is fine but it does make an assumption that the Arithmetic Logic Unit (ALU) within the CPU is perfect for handling all kinds of data. And this is not always true. There are two basic ways of doing calculations within a CPU

That is integer maths which only deal with whole numbers

And floating point maths which can deal with decimal or fractional numbers.

Large-number ranges are best handled as 'floating-point' numbers which was discussed previously. Handling floating point numbers efficiently requires wide registers to deal with a calculation in one go. And the CPU architect may not want to dedicate precious hardware space in his CPU for these wider registers.

So the idea of a 'Maths co-processor' came about. A co-processor is especially designed to carry out floating point calculation extremely quickly. It co-exists with the CPU on the motherboard. Whenever a floating point calculation needs to be done, the CPU hands the task over to the co-processor and then carries on with doing something else until the task is complete.

The advantage of having a co-processor is that calculation (and hence performance) is much faster. The disadvantage is that it is much more expensive, requires more motherboard space and takes more power.

But if the computer is dedicated to handling heavy floating point work then it may be worth it. **For instance a computer within a signal processing card in a communication system may include a maths co-processor to process the incoming data as quickly as possible.**