Syllabus Content:

3.3.5 RISC processors

- show understanding of the differences between RISC and CISC processors
- show understanding of the importance/use of pipelining and registers in RISC processors
- show understanding of interrupt handling on CISC and RISC processors

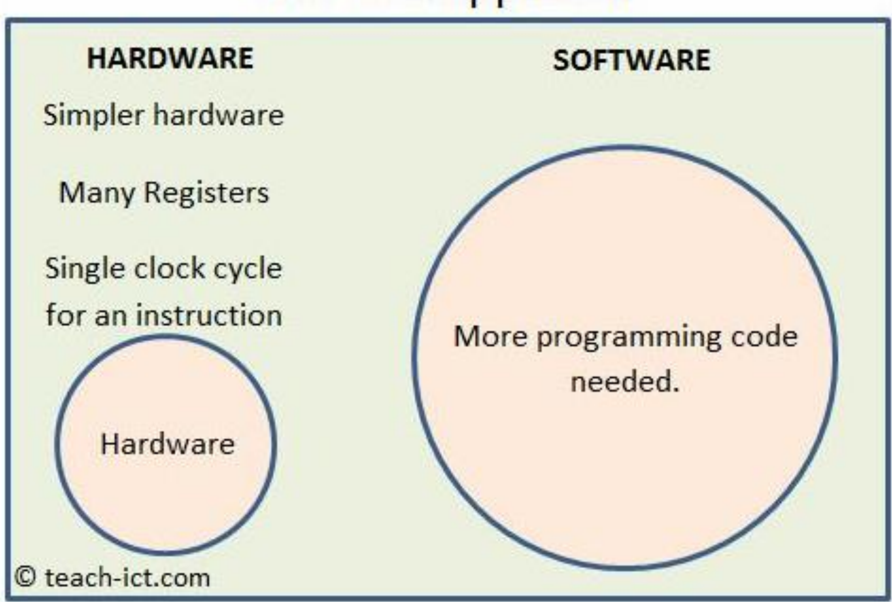
3.3.6 Parallel processing

- show awareness of the four basic computer architectures: SISD, SIMD, MISD, MIMD
- show awareness of the characteristics of massively parallel computers

3.3.5 RISC Processors:

Background to RISC

Short for **Reduced Instruction Set Computing**. The diagram below shows the RISC approach taken to processor design.



Reduced Instruction Set Computer **RISC** processors were first developed in early 1980s. Their development however has been slowed by:

- the lack of software which is written to run on RISC processor
- Intel's dominance in PC computer market using the family of x86 processors

The home is likely to have many **RISC-based** processor devices. This includes **Nintendo Wii, Microsoft Xbox 360, Sony PlayStation 3, Nintendo DS** and many **televisions** and **smartphones**.

However the desktop PC is likely to have a non-RISC processor. The reason of this is that moving to a new RISC instruction set in the processor would mean that all the existing software would no longer work.

The assembly language we studies earlier assumed that processor architecture has only one general-purpose register, the **Accumulator**. This is simplified scenario as PC processors generally have around have eight general-purpose registers.

The number of basic machine instructions would be several hundred. These computers were known as **Complex Instruction Set Computers** (**CISC**). The large numbers of instructions were matched closely to the hardware of the processor and the structures used in high level language program code.

CISC and RISC processors

The 'architecture' of a processor can be defined in a number of ways. From the point of view of a sophisticated programmer, the architecture involves the following:

- the instruction set
- the instruction format
- the addressing modes
- the registers accessible by instructions.

The choice of the instruction set is the main factor in deciding on a suitable architecture. One view is that the instruction set should be chosen so that it can be clearly applied to important problems, that only simple equipment is required and that important problems are handled speedily.

The CISC approach

HARDWARE

Complex hardware

Fewer Registers

Several clock cycles for some instructions

SOFTWARE

Less programming code

© teach-ict.com

An opposing view is that it should be chosen to suit the needs of high-level languages.

www.majidtahir.com by Majid Tahir

Early developments in computing led to the latter view becoming dominant. Computer systems contained what would now be referred to as CISC (Complex Instruction Set Computers) processors with the complexity increasing with the advent of new systems.

However, the philosophy began to be challenged in the late 1970s. It was argued that RISC (Reduced Instruction Set Computers) would be a better approach. **Table below** contains a number of features that distinguish RISC from CISC.

It can be seen that 'reduced' affects more than just the number of instructions.

The simplicity of the instructions allows data to be stored in registers and manipulated in them with no resource to memory access other than that necessary for initial loading and possible final storing. The simplicity also allows hard-wiring inside the control unit with limited complexity required.

Comparison of CISC and RISC

RISC	CISC
Fewer instructions (Each instruction takes exactly one clock cycle)	More instructions (Clock cycles taken by instruction may vary)
Simpler instructions	Complex instructions
Small number of instruction formats	Many instruction formats
Single-cycle instructions whenever possible	Multi-cycle instructions
Fixed-length instructions	Variable-length instructions
Only load and store instructions to address memory	Many types of instructions to address memory
Fewer addressing modes	More addressing modes
Multiple register sets (Large number of General-purpose registers)	Fewer registers (Limited number of General-purpose registers)
Hard-wired control unit	Micro-programmed control unit
Pipelining easier	Pipelining more difficult
Instructions and data held in **RAM**	Extensive use made of **Cache Memory**
The design emphasis is on the software	The design emphasis is on the hardware
Processor chips require fewer transistors	Uses the memory unit to allow complex instructions to be carried out

www.majidtahir.com by Majid Tahir

In contrast, the specialised instructions that can be part of **CISC** architecture often require repeated memory access.

The complexity of some of the instructions makes hard-wiring extremely difficult so microprogramming is the norm.

Hardwired control unit: Hardwired control units are implemented through use of combinational logic units, featuring a finite number of gates that can generate specific results based on the instructions that were used to invoke those responses. No ROM inside CU and no microprogramming available. Hardwired control units are generally faster than **micro-programmed** designs.

A comparison of two processors is shown in Table below

	Intel 80486 (CISC)	Sun SPARC (RISC)
Number of instructions	235	69
Instruction size (bytes)	Between 1 and 11	4
Addressing modes	11	1
General-purpose registers	8	520

However, the increased complexity of instructions for CISC is often because they more closely match high-level language constructs. This means that compiler writing becomes much easier for a CISC processor.

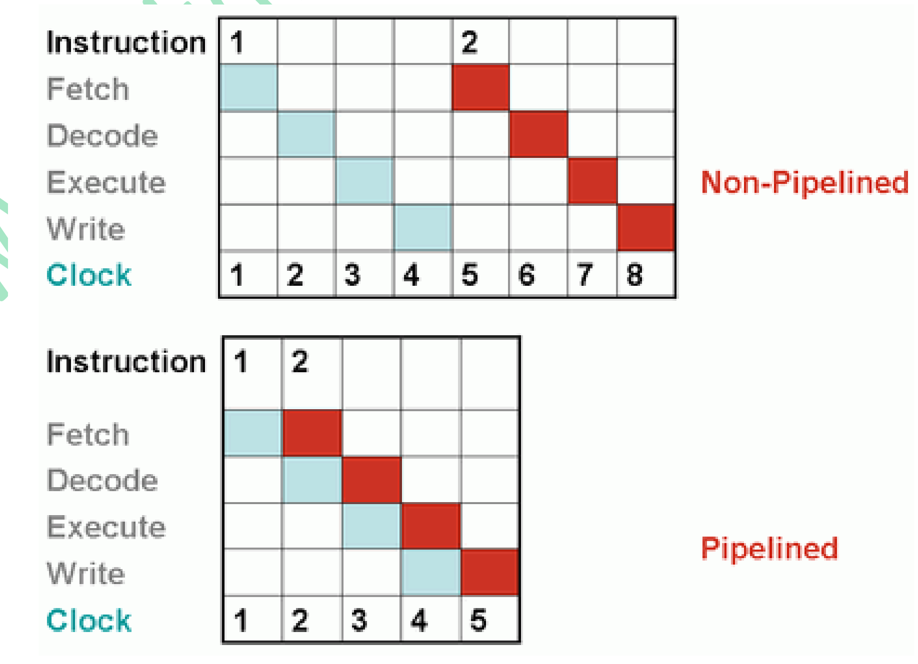
One of the major driving forces for creating RISC processors was the opportunity they would provide for efficient pipelining.

Pipelining:

Pipelining is an implementation technique where multiple instructions are overlapped in execution.

Pipelining is a technique used to improve the execution throughput of a CPU by using the processor resources in a more efficient manner.
The basic idea is to split the processor instructions into a series of small independent stages. Each stage is designed to perform a certain part of the instruction. At a very basic level, these stages can be broken down into:

- **(IF) Instruction Fetch:** Fetches an instruction from memory
- **(ID) Instruction Decode:** Decodes the instruction be executed
- **(IE) Instruction Execute:** Executes the instruction
- **(IW) Instruction Write:** Writes the result back to register or memory



There will be a dedicated CPU module for each of the stages mentioned above. **Blue boxes** are for 1st instruction and **Red boxes** are for second instructions.

In a non-pipelined CPU, when an instruction is being processed at a particular stage, the other stages are at an idle state – which is very inefficient. If you look at the diagram, when the 1st instruction is being decoded, the Fetch, Execute and Write Units of the CPU are not being used and it takes 8 clock cycles to execute the 2 instructions.

The underlying principle of pipelining is that the fetch-decode-execute cycle can be separated into a number of stages. One possibility is a five-stage model consisting of:

1. instruction fetch (IF)
2. instruction decode (ID)
3. operand fetch (OF)
4. instruction execute (IE)
5. result write back (WB).

Clock cycles

Processor stages	1	2	3	4	5	6	7	8	9	10
IF	A	B	C	D	E	F				
ID		A	B	C	D	E	F			
OF			A	B	C	D	E	F		
IE				A	B	C	D	E	F	
WB					A	B	C	D	E	F

To demonstrate how pipelining works, we will consider a program which has six instructions **(A, B, C, D, E and F). Figure above** shows the relationship between processor stages and the number of required clock cycles when using pipelining. It shows how pipelining would be implemented with each stage requiring one clock cycle to complete.

This functionality clearly requires processors with several registers to store each of the stages.

- Execution of an instruction is split into a number of stages; as each stage completes, the first stage of the first instruction can now be executed.
- Then the second instruction can start execution before the first one has completed, and so on, until all six instructions are processed.
- In this example, by the time instruction 'A' has completed, instruction 'F' is at the first stage and instructions 'B' to 'E' are at various in-between stages in the process. As Figure shows, a number of instructions can be processed at the same time, and there is no need to wait for an instruction to go through all five cycles before the next one can be implemented.
- In the example shown, the six instructions require 10 clock cycles to go to completion. Without pipelining, it would require 30 (6 × 5) cycles to complete (since each of the six instructions requires five stages for completion).

One issue that has to be dealt with regarding a pipelined processor is interrupt handling. The discussion in Chapter 5 referred to a processor with instructions handled sequentially. In the pipelined system described above there will be five instructions in the pipeline when an interrupt occurs. One option for handling the interrupt is to erase the pipeline contents for the latest four instructions to have entered. Then the normal interrupt-handling routine can be applied to the remaining instruction.

The other option is to construct the individual units in the processor with individual program counter registers. This allows current data to be stored for all of the instructions in the pipeline while the interrupt is handled.

Problems with pipelining:

Two issues will cause the pipeline to stall:

- dealing with a data dependency between instructions
- branch instructions.

Data dependency:

Progress check

A program contains the following sequence of instructions:

//add the contents of R1 and R2;
store the result in R3

ADD R3, R1, R2
ADD R5, R4, R3

Explain the data dependency
here……………………………………………………………………………………..

………………………………………………………………………………………………………
…………………………

………………………………………………………………………………………………………
…………………………

The detail of pipelining of the two istructions is as follows:

- When instruction 2 is at its Decode process, the processor will read the value of R3 and R4
- Instruction 1 is one step ahead, so at this time the contents of R1 and R2 are being added, but will not yet have been written to R3
- Therefore the second instruction is unable to read the R3 value it needs.
- The pipeline is stalled and loaded with a number of empty instructions called **‘bubbles’**

www.majidtahir.com

One technique for dealing with data-dependent instructions is for them to be identified by the compiler which will then attempt to re-order the instructions.

Branch instructions:

Consider the following sequence of instructions

Loop: ADD R3, R2, R1 // add R1 to R2 and store in R3

ADD R6, R5, R4 // add R4 to R5 and store in R6

JPE R3, R6, LOOP // compare R3 and R6 –if equal jump to address LOOP

The issue is same here as Progress check, the third instruction has to know the values in Register R3 and R6. These are not known as neither instruction 1 or instruction 2 has yet written the value to the register. This can cause the pipeline to stall.

One strategy that pipelining can use to deal with this branch prediction. The processor makes a guess at the outcome of the condition Research has shown that if the branch instruction is at the bottom of the loop, the execution will go back to the start of the loop in around 90% cases. Conditions at the start of loop are true in 50% cases. Therefore the strategy is to assume the condition is true in the first case and not true in the second case. If the guess proves to be wrong then the processor must re-instate the register contents and start the pipeline again with correct instructions.

Interrupt handling on CISC & RISC processors:

The use of interrupt on a RISC processor is no different. The same definition of an interrupt holds: a signal sent to the processor from hardware device to indicate that device needs attention.

The use of interrupts avoids the processor having to regularly check to see if a hardware device needs its attention. This strategy (called pooling) was in use before interrupts.

The system uses vectored interrupts. Every device is assigned a device number that corresponds to bits sent to an interrupt register. Hence from the number, the processor knows the source of interrupt.

Once the interrupt is received, the state of all registers must be saved and the appropriate interrupt service routine (ISR) code executed.

Parallel processing:

Parallel processing means that the architecture has more than one processor. Different processors are responsible for different parts of tasks. The programmer must design the code so that specific code is used for processing of the task’s component parts. Each task is then processed by different processor. The software will integrate the data produced to provide the final software solution

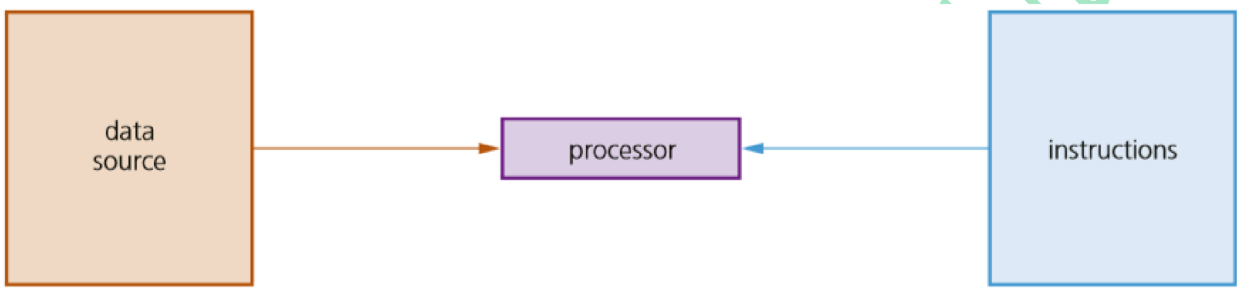
Four variants of parallel processing are:

www.majidtahir.com by Majid Tahir

- SISD Single Instruction Single Data
- SIMD Single Instruction Multiple Data
- MISD Multiple Instruction Single Data
- MIMD Multiple Instruction Multiple Data

SISD (Single Instruction Single Data stream):

SISD (Single Instruction Single Data stream) is the typical arrangement found in early personal computers. There is a single processor with one data source which works on a single algorithm, so no processor parallelism. The single data stream just means one memory.
Each task is processed in a sequential order. Since there is a single processor, this architecture does not allow for parallel processing.

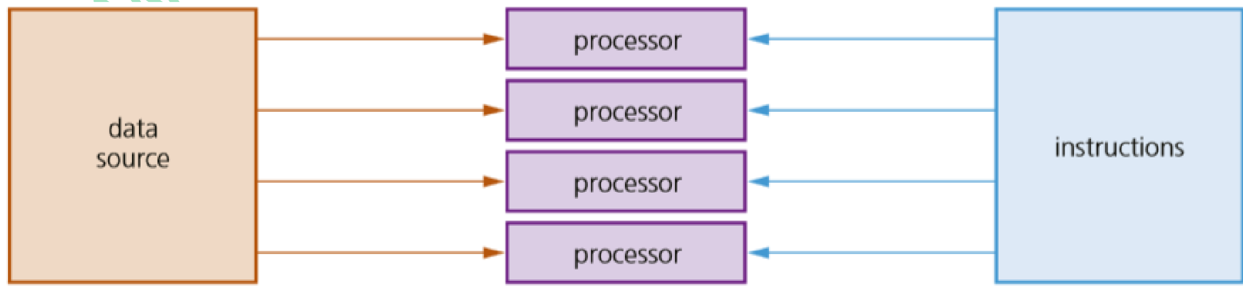


SIMD (Single Instruction Multiple Data stream):

SIMD (Single Instruction Multiple Data stream) describes how an array or vector processor works.
SIMD (single instruction multiple data) uses many processors. Each processor executes the same instruction but uses different data inputs – they are all doing the same calculations but on different data at the same time.

SIMD are often referred to as array processors; they have a particular application in graphics cards. For example, suppose the brightness of an image made up of 4000 pixels needs to be increased. Since SIMD can work on many data items at the same time, 4000 small processors (one per pixel) can each alter the brightness of each pixel by the same amount at the same time.

This means the whole of the image will have its brightness increased consistently. Other applications include sound sampling – or any application where a large number of items need to be altered by the same amount (since each processor is doing the same calculation on each data item).



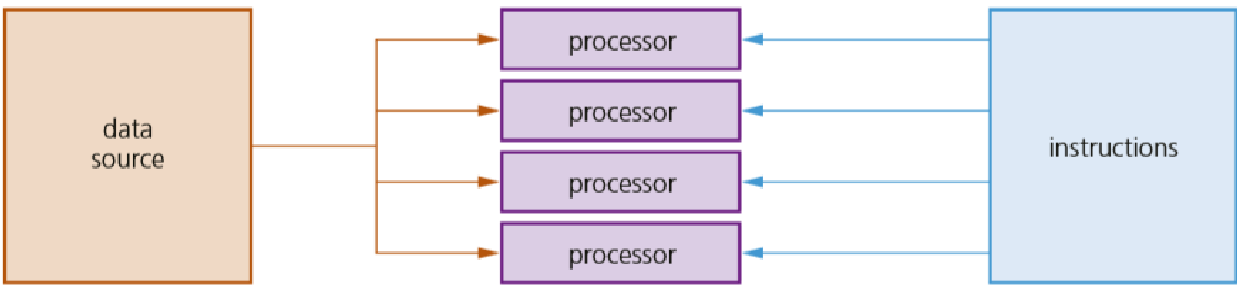
This is an appropriate architecture for problems which need to do an analysis of the large dataset using the same criteria. The several processors each have its own local cache memory. This

www.majidtahir.com

makes possible a single program instruction which performs the same action simultaneously on several data items.

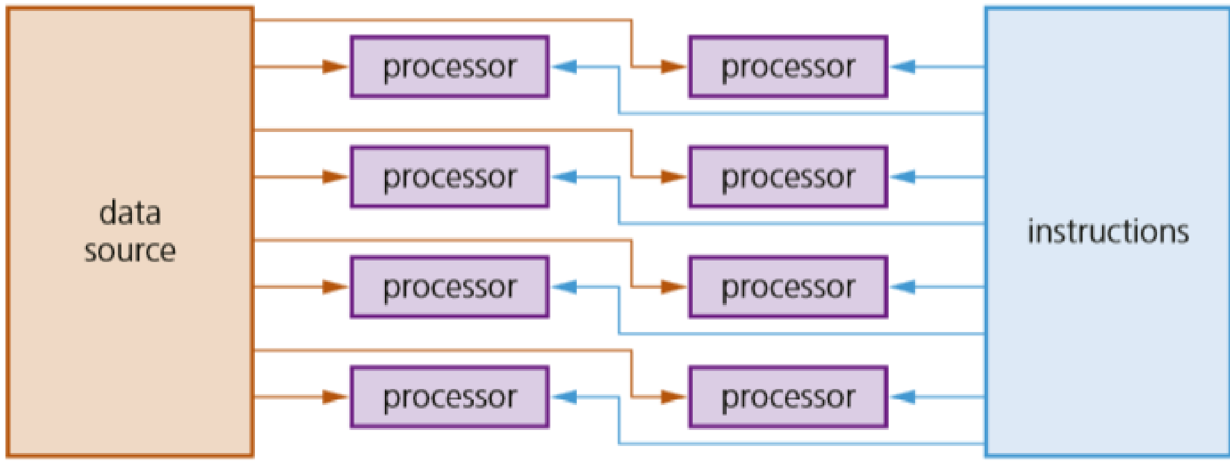
MISD (Multiple Instruction Single Data stream):

Many processors perform operations on same data value. The data may be one value from an array. One strategy for MISD is the parallel input of data value through a network of processor nodes. The nodes (whose behavior is programmable with software) will merge or sort the data values into final result.
MISD (multiple instruction single data) uses several processors. Each processor uses different instructions but uses the same shared data source. MISD is not a commonly used architecture (MIMD tends to be used instead). However, the American Space Shuttle flight control system did make use of MISD processors



MIMD (Multiple Instruction Multiple Data stream):

MIMD (multiple instruction multiple data) uses multiple processors. Each one can take its instructions independently, and each processor can use data from a separate data source (the data source may be a single memory unit which has been suitably partitioned). The MIMD architecture is used in multicore systems (for example, by super computers or in the architecture of multi-core chips).



MIMD has examples in modern personal computers which are of the symmetric multiprocessor type using identical processors. In this case, each processor executes a different individual instruction. The multiple data stream can be provided by a single memory suitably partitioned. Each processor might have a dedicated cache memory.

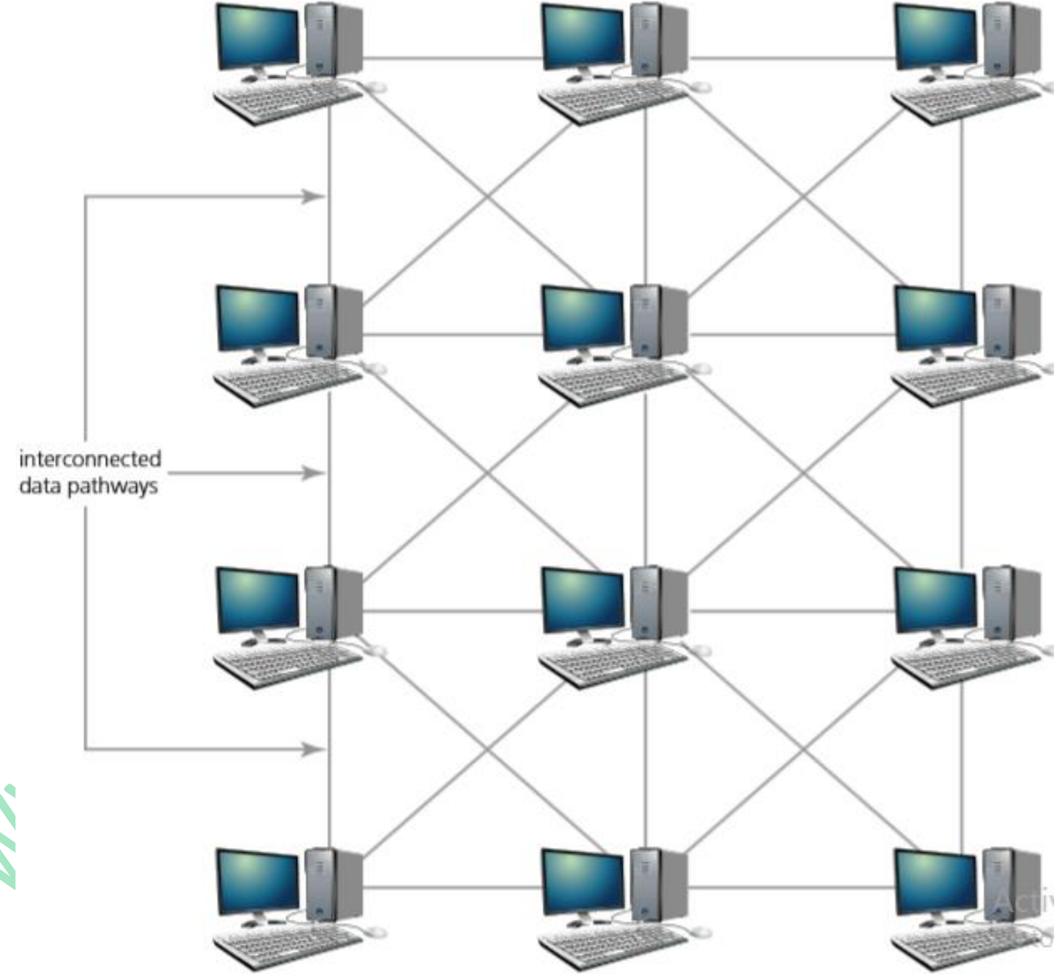
Characteristics of massively parallel computers

Parallel computer systems Examples of one type of multicomputer system are called massively parallel computers. These are the systems used by large organisations for computations involving highly complex mathematical processing. They are the latest in an evolution of what have traditionally been called **'supercomputers'.**

Massively parallel computers

The major difference in architecture is that instead of having a bus structure to support multiple processors there is a network infrastructure to support multiple computer units. The programs running on the different computers can communicate by passing messages using the network.

An alternative type of multicomputer system is cluster computing, where a very large number of PCs are networked.



The processor from each computer forms part of a larger pseudo-parallel system which can act like a super computer. Some textbooks and websites also refer to this as grid computing. Massively parallel computers have evolved from the linking together of a number of computers,

effectively forming one machine with several thousand processors. This was driven by the need to solve increasingly complex problems in the world of science and mathematics.

By linking computers (processors) together in this way, it massively increases the processing power of the 'single machine'. This is subtly different to cluster computers where each computer (processor) remains largely independent. In massively parallel computers, each processor will carry out part of the processing and communication between computers is achieved via interconnected data pathways. Figure shows this simply.

Exam-style Questions

1 (a) Computer systems are now often constructed with RISC processors.

(i) State what the acronym RISC stands for. [1]
(ii) State four characteristics to be expected of a RISC system. [4]

(b) A RISC processor is likely to be 'hard-wired'.

i. Explain what this term means and which specific part of the processor will be hard-wired. [3]
ii. State what the alternative to hard-wiring is and what hardware component is needed to be part of the processor to allow this alternative to be implemented. [2]

2 (a) Parallelism can be achieved in a number of ways.

i. Identify three different types of parallelism. [3]
ii. Identify which type pipelining belongs to. [1]
iii. Using a diagram, explain how pipelining works. [5]

b. Interrupt handling is not so straightforward in a pipelined system. Explain why this is so and give a brief account of how problems can be avoided. [3]

www.majidtahir.com by Majid Tahir

Progress answer:

The second instruction needs the value in register R3. If instruction1 has not completed, this value will not be available to instruction 2.

Answer of exam style questions:

Exam-style Questions (with mark allocation in brackets)

1 a i Reduced Instruction Set Computer (1).

ii Any of those listed in Table 19.01 of the coursebook (1 each, max 4).

b i Relates to the internal workings of the control unit (1), logic circuits (1) are used to handle instructions (1), no ROM inside the CU (1), no microprogramming (1). (Max 3)

ii Microprogramming (1), firmware (1), a ROM inside the CU (1). (Max 2 ; no mark given if the same point has been made in answer to b i).

2 a i Instructions (1), processors (1), memory usage (1) computer systems (1). (Max 3)

ii Instructions (1).

iii A diagram as shown in Figure 19.01 in the coursebook (2), stages in instruction handling described (3), comment on progress through diagram (1). (Max 5)

b Interrupts normally detected and handled when an instruction has completed execution (1), if in a pipelined system one instruction has completed while others have not (1) interrupt handling routine has to be put into the pipeline (1), might erase other instructions (1) might have dedicated registers for each strand of the pipeline (1). (Max 3)

References:

- Computer Science Course Book by Sylvia Langfield & Dave Duddell
- Cambridge international AS & A level by david Watson & Hellen Williams (Hodder education)
- AS & A level Computer Science Teacher's resource CD
- Computer Science Revision Guide by Tony Piper
- Teacher Support Guide (CIE Resource)
- https://whatis.techtarget.com/definition/pipelining
- http://web.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/pipe_title.html
- https://stackpointer.io/hardware/how-pipelining-improves-cpu-performance/113/