BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 3

DCA2103

COMPUTER ORGANIZATION

Unit 14

Parallel Organization

Table of Contents

1. INTRODUCTION

In the previous unit, we discussed about the concepts of microprogramming, Computer Clock, Microinstructions and its timing, control path, microcode, and machine instructions. This unit presents the concept called parallel organization. We also study in this unit the modern processor technology. So we introduce instruction-set architectures (ISA) including CISC and RISC. We also discuss the basic concepts of vector processing and the necessary implementation requirements. We distinguish vector processing from scalar processing, present the characteristics of vector instructions, and define the performance measures of vector processors.

1.1 Objectives

After studying this unit, you should be able to:

- *Define parallel organization*
- *Define UMA and NUMA.*
- *List the characteristics of RISC and CISC processors*
- *List the characteristics of vector processing*
- *Explain superscalar processors*
- *Discuss the superscalar issues.*

2. PAEALLEL ORGANIZATION

In a parallel organization, multiple processing units cooperate to execute applications. A parallel processing organization looks for a grosser level of parallelism, one that enables work to be done in parallel, and cooperatively, by multiple processors. A number of issues are raised by such organizations. For example, if multiple processors, each with its own cache, share access to the same memory, hardware or software mechanisms must be employed to ensure that both processors share a valid image of the main memory. This is known as the cache coherence problem.

Flynn proposed the following categories of computer systems:

Single instruction, single data (SISD) stream: A single processor executes a single instruction stream to operate on data stored in a single memory.

Single instruction, multiple data (SIMD) stream: A single machine instruction controls the simultaneous execution of a number of processing elements on a lockstep basis. Each processing element has an associated data memory so that each instruction is executed on a different set of data by the different processors.

Multiple instruction, single data (MISD) stream: A sequence of data is transmitted to a set of processors, each of which executes a different instruction sequence.

Multiple instructions, multiple data (MIMD) stream: A set of processors simultaneously execute different instruction sequences on different data sets. SMPs, clusters, and NUMA systems fit into this category.

With the MIMD organization, the processors are general purpose; each is able to process all of the instructions necessary to perform the appropriate data transformation. If the processors share a common memory, then each processor accesses programs and data stored in the shared memory, and processors communicate with each other via that memory. The most common form of such a system is known as a **symmetric multiprocessor (SMP).** Here in an SMP, multiple processors share a single memory or pool of memory by means of a shared bus or other interconnection mechanisms; a distinguishing feature is that the memory access time to any region of memory is approximately the same for each processor. A more recent development is **the non-uniform memory access (NUMA)** organization. As

the name suggests, the memory access time to different regions of memory may differ for a NUMA processor. A collection of independent uniprocessors or SMPs may be interconnected to form a **cluster.**

The design issues relating to SMPs, clusters, and NUMAs are complex, involving issues relating to physical organization, interconnection structures, interprocessor communication, operating system design, and application software techniques.

Symmetric multiprocessor (SMP)

The term *SMP* refers to a computer hardware architecture and also to the operating system behavior that reflects that architecture. An SMP can be defined as a standalone computer system with the following characteristics:

1. There are two or more similar processors of comparable capability.
2. These processors share the same main memory and I/O facilities and are interconnected by a bus or other internal connection scheme, such that memory access time is approximately the same for each processor.
3. All processors share access to I/O devices, either through the same channels or through different channels that provide paths to the same device.
4. All processors can perform the same functions (hence the term *symmetric*).
5. The system is controlled by an integrated operating system that provides interaction between processors and their programs at the job, task, file, and data element levels.

The operating system of an SMP schedules processes or threads across all of the processors. An attractive feature of an SMP is that the existence of multiple processors is transparent to the user. The operating system takes care of the scheduling of threads or processes on individual processors and of synchronization among processors. The most common organization for personal computers, workstations, and servers is the time-shared bus. The time-shared bus is the simplest mechanism for constructing a multiprocessor system. Figure 1 shows the organization of the Symmetric Multiprocessor.

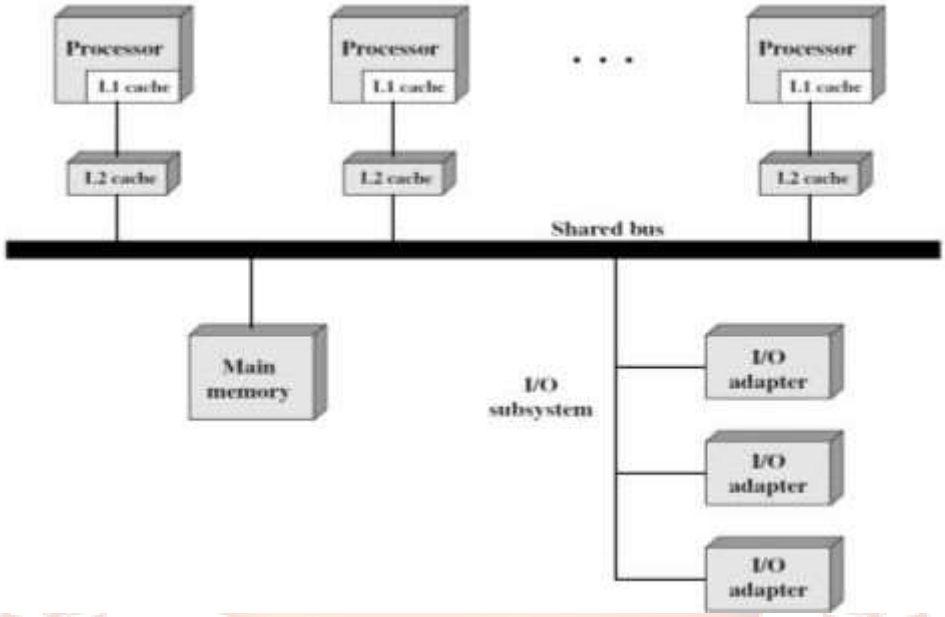


Figure 1: Organization of Symmetric Multiprocessor.

Clusters

Clustering is an alternative to symmetric multiprocessing as an approach to providing high performance and high availability and is particularly attractive for server applications. We can define a cluster as a group of interconnected, whole computers working together as a unified computing resource that can create the illusion of being one machine. Here the term *whole computer* means a system that can run on its own, apart from the cluster.

Cluster Configurations

Clusters are classified based on whether the computers in a cluster share access to the same disks. Figure 14.2a shows a two-node cluster in which the only interconnection is by means of a high-speed link that can be used for message exchange to coordinate cluster activity.

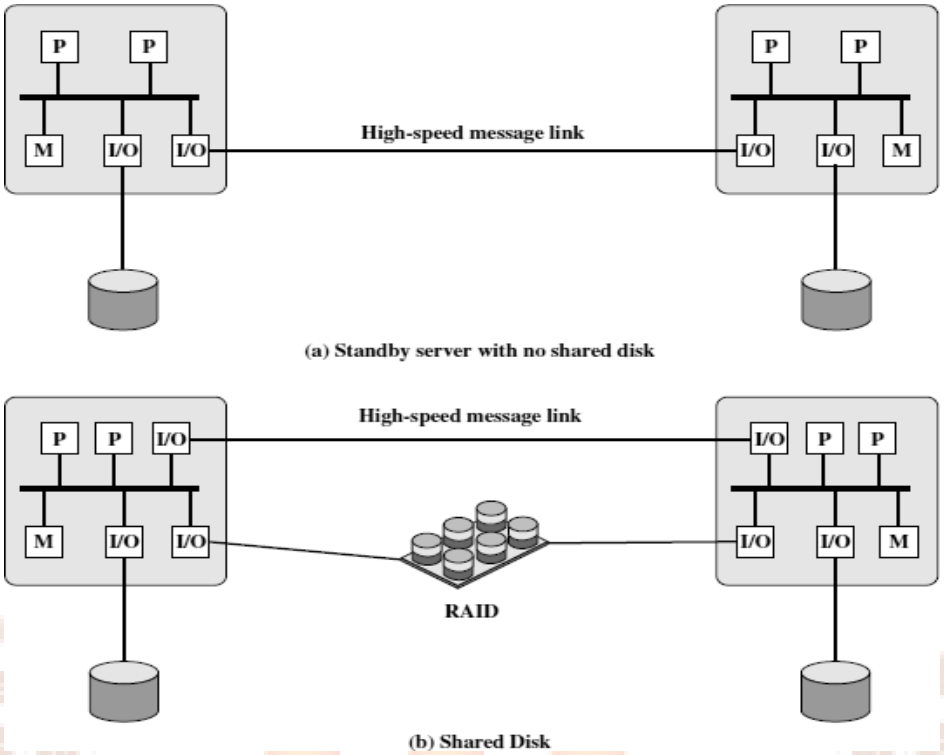


Figure 2: Cluster Configurations

Figure 2(b) shows the other alternative called a shared-disk cluster. Here generally there is a message link between nodes. Also, there is a disk subsystem that is directly linked to multiple computers within the cluster. The common disk subsystem is a RAID system. The use of RAID or some similar redundant disk technology is common in clusters so that the high availability achieved by the presence of multiple computers is not compromised by a shared disk that is a single point of failure.

Uniform Memory Access (UMA)

UMA is a shared memory architecture used in parallel computers. All the processors in the UMA model share the physical memory uniformly. In a UMA architecture, access time to a memory location is independent of which processor makes the request or which memory chip contains the transferred data. Uniform Memory Access computer architectures are often contrasted with Non-Uniform Memory Access (NUMA) architectures. In the UMA architecture, each processor may use a private cache. Peripherals are also shared in some fashion; The UMA model is suitable for general purpose and time-sharing applications by multiple users. It can be used to speed up the execution of a single large program in time critical applications.

Non-Uniform Memory Access (NUMA)

NUMA is a computer memory design used in multiprocessing, where the memory access time depends on the memory location relative to a processor. A processor can access its own local memory faster than non-local memory, that is, memory local to another processor or memory shared between processors. NUMA architectures logically follow in scaling from symmetric multiprocessing (SMP) architectures. A NUMA system without cache coherence is more or less equivalent to a cluster.

3. INSTRUCTION SET ARCHITECTURE (ISA)

The instruction set of a computer specifies the primitive commands or machine instructions that a programmer can use in programming. The complexity of instruction is attributed to the instruction formats, data format, addressing modes, general-purpose registers, op-code specifications, and flow control mechanism used.

3.1 RISC and CISC

The world of Microprocessors and CPUs can be divided into two parts:

- Complex instruction set computers using CISC processors and
- Reduced instruction set computers with RISC processors.

A Reduced Instruction Set Computer (RISC) is a microprocessor that has been designed to perform a small set of instructions, with the aim of reducing the overall speed of the processor.

A Complex Instruction Set Computer (CISC) is a processor designed with complete computer instruction set to provide maximum capabilities in the most efficient way. These support high-level languages instructions and also allow complex addressing modes for array and data structures in single instructions. These processors can execute several low-level operations like arithmetic operations, fetch from memory, store memory, etc., and various multi-step instructions and complex addressing modes in fewer instructions. As their names imply, CISC and RISC differ in the complexities of their instruction sets. CISC processors have larger instruction sets that often include some particular complex instructions. These instructions usually correspond to a specific statement in high-level languages. Intel's Pentium class microprocessors fall into this category. In contrast, RISC processors exclude these instructions, opting for smaller instruction sets with simpler instructions. The RISC architecture involves an attempt to reduce execution time by simplifying the instruction set of the computer.

3.2 Characteristics of CISC & RISC

Major characteristics of CISC architecture are

1) A large number of instructions - typically from 100 to 250 instructions

2) Some instructions perform a specialized task and are used infrequently.
3) A large variety of addressing modes – typically from 5 to 20 different modes.
4) Variable length instruction formats.
5) Instructions that manipulate, operands in memory.

Major Characteristics of RISC architecture are

1) Relatively few instructions.
2) Relatively few addressing modes.
3) Memory access is limited to load and store instructions.
4) All operations are done within the registers of the CPU.
5) Fixed length, easily decoded instruction format.
6) Single-cycle instruction execution.
7) Hard wired rather than micro-programmed control.

Other characteristics attributed to RISC architecture are:

1) The relatively large number of registers in the processor unit.
2) Use of overlapped register windows to speed-up procedure call and return.
3) Efficient instruction pipeline.
4) Compiler support for efficient translation of high-level language programs into machine language programs.

3.4 RISC Vs CISC

Given the differences between RISC and CISC, Which is better? The answer to this question is that there is no right answer to this question. Each has some features that are better than the other.

RISC processors have fewer and simpler instructions than CISC processors. As a result, their control units are less complex and easier to design. This allows them to run at higher clock frequencies than CISC processors and reduces the amount of space needed on the processor chip, so the designer can use the extra space for additional registers and other components. Simpler control units can also lead to reduced development costs. With a simpler design, it is easier to incorporate parallelism into the CU of the RISC CPU.

With fewer instructions in their instruction sets, the compilers for RISC processors are less complex than those for CISC processors. As a general guideline, CISC processors were originally designed for assembly language programming, whereas RISC processors are geared toward compiled, high-level–language programs. However, the same complied high-level program will require more instructions for a RISC CPU than for a CISC CPU. The CISC methodology offers some advantages as well. Although CISC processors are more complex, this complexity does not necessarily increase development costs. Current CISC processors are often the most recent addition to an entire family of processors, such as Intel's family. As such they might incorporate portions of the designs of their previous families.

CISC processors also provide backward compatibility with other processors in their families. If they are pin-compatible, it may be possible simply to replace a previous generation processor with the newest model without changing the rest of the computer's design. This same backward compatibility, whether pin compatible or not, allows the CISC CPU to run the same software as used by the predecessors in its family. For instance, a program that runs successfully on a Pentium III. This can translate into significant savings for the user and can determine the success or failure of a microprocessor.

CISC designs generally incorporate instruction pipelines, which have improved performance dramatically in RISC processors. As technology allows more devices to be incorporated into a single microprocessor chip, CISC is adding more registers to their designs, again to achieve the performance improvements they provide to RISC processors. Newer processor families such as PowerPC microprocessors, draw some features from RISC methodology and others from CISC, Making them a hybrid of RISC and CISC.

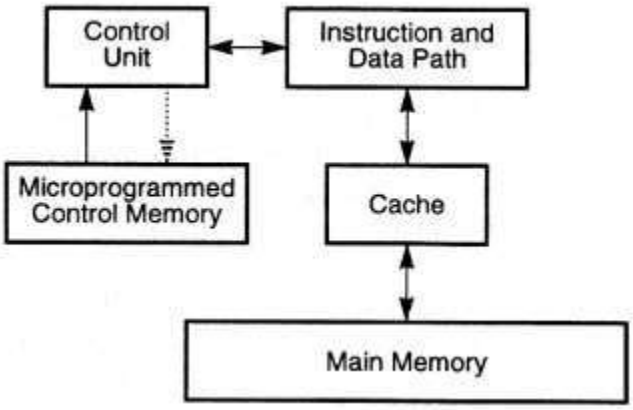
3.5 Instruction Set Complexity: RISC vs. CISC

The primary objective of processor designers is to improve performance. Performance is defined as the amount of work that the processor can do in a given period of time. Different instructions perform different amounts of work.

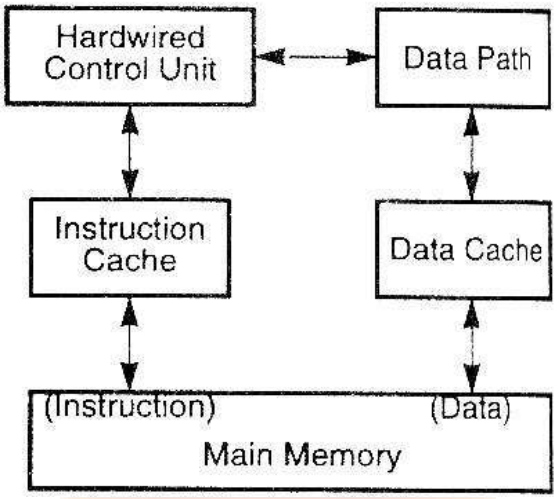
To increase performance, you can either have the processor execute instructions in less time or make each instruction it executes do more work. Increasing performance by executing

instructions in less time means that, increasing the clock speed of the processor. Making it do more work with each instruction means increasing the power and complexity of each instruction. A real-life analogy would be to imagine you pedaling a bicycle. To get where you are going more quickly, you can either use a low gear and pedal very quickly or use a high gear and push harder. You can try to push both harder and faster, but you can never pedal as fast with a high gear as you can with a low one.

This trade-off in basic instruction set design philosophy is reflected in the two main labels given to *instruction* sets. CISC stands for complex instruction set computer and is the name given to processors that use a large number of complicated instructions, to try to do more work with each one. *RISC* stands for *reduced instruction set computer* and is the generic name given to processors that use a small number of simple instructions, to try to do less work with each instruction but execute them much faster. The question of which of these two approaches to use in designing a processor has become one of the great arguments of the computer world. This is especially true because once a platform makes an instruction set decision; it tends to stick with it in order to ensure compatibility with existing software. Hardware features built into CISC and RISC processors are compared here. Figure 3(a) and 3(b) show the architectural distinctions between modern CISC and traditional RISC. Future processors may be designed with features from both types.



(a) CISC architecture



(b) RISC architecture

Figure 3: Architectural distinctions

(a) CISC architecture (b) RISC architecture

Conventional CISC architecture uses a unified cache for holding both instructions and data. Hence they must share the same data/instruction path. In a RISC processor, separate instruction and data cache are used with different access paths. In other words, CISC processors may also use split codes. Traditional CISC use micro-programmed control and most RISC processors use hardwired control. Thus control memory (ROM) is needed in earlier CISC processors, which may significantly slow down the instruction execution. Modern CISC may also use hardwired control. Therefore split caches and hardwired control are not exclusive to RISC machines. The comparison of RISC and CISC processors' main features with respect to five major areas specifically is given in table 1.

Table 1: Comparison of RISC and CISC architectural characteristics

Architectural Characteristic	Complex Instruction Set Computer (CISC)	Reduced Instruction Set Computer (RISC)
Instruction-set size and instruction formats	Large set of instructions with variable formats (16–64 bits per instruction).	Small set of instructions with fixed (32-bit) format and most register-based instructions.
Addressing modes	12–24.	Limited to 3–5.
General-purpose registers and cache design	8–24 GPRs, mostly with a unified cache for instructions and data, recent designs also use split caches.	Large numbers (32–192) of GPRs with mostly split data cache and instruction cache.
Clock rate and CPI	33–50 MHz in 1992 with a CPI between 2 and 15.	50–150 MHz in 1993 with one cycle for almost all instructions and an average CPI < 1.5.
CPU Control	Most microcoded using control memory (ROM), but modern CISC also uses hardwired control.	Most hardwired without control memory.

Self-Assessment Questions – 1

1. processors have larger instruction sets that ofteninclude some particular complex instructions.
2. The_____of RISC are less complex and easier to design.
3. Most RISC processors use_____- control.

4. VECTOR PROCESSING REQUIREMENTS

Scalar processors are those executing one instruction per cycle. That is only one instruction is issued per cycle, and only one compilation of instruction is expected from the pipeline per cycle. A base scalar processor is defined as a machine with one instruction is issued per cycle, one cycle latency for a simple operation, and one cycle latency between instruction issues. Where instruction issue latency is defined as the time in cycles required between issuing of two adjacent instructions.

A vector processor executes vector instructions on arrays of data. Here each instruction involves a string of repeated operations, which are ideal for pipelining with one result per cycle.

4.1 Characteristics of Vector Processing

A vector operand contains an ordered set of *n* elements, where *n* is called the *length* of the vector. Each element in a vector is a scalar quantity, which may be a floating-point number, an integer, a logical value, or a character (byte). Register-based vector instructions appear in most of the register-to-register vector processors like the Cray supercomputer. Let Vi denote a vector register of length n, si is a scalar register and M(1:n) is a memory array of length n. Typical Register based operations are listed below:

1) Binary vector (cross product) $V_1 \times V_2 \rightarrow V_3$
2) Scaling (scalar multiplication) $s_1 \bullet V_2 \rightarrow V_3$
3) Binary reduction (Dot product) $V_1 \circ V_2 \rightarrow s_3$
4) Memory loading $M(1{:}n) \rightarrow V_1$
5) Vector storing $V_1 \rightarrow M(1{:}n)$
6) Unary vector $V_2 \rightarrow V_3$
7) Unary reduction $\circ V_2 \rightarrow s_3$

Memory-based vector operations are found in memory-to-memory vector processors. Let Mi(1:n) be a vector of length n and M(k) denotes the quantity stored in memory location k. The following are few examples of memory based operations:

1) Vector product $M(1{:}n) \times M_2(1{:}n) \rightarrow M_3(1{:}n)$

2) Scaling $s_1 \circ M_2(1{:}n) \rightarrow M_3(1{:}n)$

and so on. (Similar to register-based operations).

Some special instructions may be used to facilitate the manipulation of vector data. A Boolean vector can be generated as a result of comparing two vectors, and can be used as a masking vector for enabling or disabling component operations in a vector instruction. A compress instruction will shorten a vector under the control of a masking vector. A merge instruction combines two vectors under the control of a masking vector. Compress and merge are special operations because the resulting operand may have a length different from that of the input operands.

In general, machine operations suitable for pipelining should have the following properties:

a. Identical processes (or functions) are repeatedly invoked many times, each of which can be subdivided into sub-processes (or sub-functions).
b. Successive operands are fed through the pipeline segments and require as few buffers and local controls as possible.
c. Operations executed by distinct pipelines should be able to share expensive resources, such as memories and buses, in the system.

These characteristics explain why most vector processors have pipeline structures. Vector instructions need to perform the same operation on different data sets repeatedly. This is not true for scalar processing over a single pair of operands. One obvious advantage of vector processing over scalar processing is the elimination of the overhead caused by the loop-control mechanism. Because of the start-up delay in a pipeline, a vector processor should perform better with longer vectors.

Vector instructions are usually specified by the following fields:

1. The *operation code* must be specified in order to select the functional unit or to reconfigure a multifunctional unit to perform the specified operation. Usually, microcode control is used to set up the required resources.

2. For a memory-reference instruction, the *base addresses* are needed for both source operands and result vectors. If the operands and results are located in the vector register file, the designated vector registers must be specified.
3. The address increment between the elements must be specified.
4. The *address offset* relative to the base address should be specified. Using the base address and the offset, the effective memory address can be calculated. The offset, either positive or negative, offers the use of skewed vectors to achieve parallel accesses.
5. The vector length is needed to determine the termination of a vector instruction. A masking vector may be used to mask off some of the elements without changing the contents of the original vectors.

We can classify pipeline vector computers in two architectural configurations according to where the operands are retrieved. In a Vector processor, one class is the *memory-to-memory* architecture, in which source operands, intermediate and final results are retrieved directly from the main memory. For memory-to-memory vector instructions, the information of the base address, the offset, the increment, and the vector length must be specified in order to enable streams of data transfers between the main memory and the pipelines.

The other class has a *register-to-register* architecture, in which operands and results are retrieved indirectly from the main memory through the use of a large number of vectors or scalar registers. The vector length affects the processing efficiency because of the additional overhead caused by subdividing a long vector. In order to enhance the vector-processing capability, an optimized object code must be produced to maximize the utilization of pipeline resources. The following approaches have been suggested:

Enrich the vector instruction set: With a richer instruction set, the processing capability will be enhanced. One can avoid excessive memory access and poor resource utilization with an improved instruction set.

Combine scalar instructions: Using a pipeline for processing scalar quantities, one should group scalar instructions of the same type together as a batch instead of interleaving them. The overhead due to the pipeline reconfiguration can be greatly reduced by grouping scalar instructions.

Choose suitable algorithms: Often a fast algorithm that is implemented in a serial processor may not be at all effective in a pipelined processor. For example, the merge-sort algorithm is more suitable for pipelining because the machine can merge two ordered vectors in one pass.

Use a vectorizing compiler: An intelligent compiler must be developed to detect the concurrency among vector instructions, which can be realized with pipelining or with the chaining of pipelines. A vectorizing compiler would regenerate parallelism lost in the use of sequential languages. It is desirable to use high-Level programming languages with rich parallel constructs on vector processors. The following four stages have been recognized in the development of parallelism in advanced programming. The parameter in parentheses indicates the degree of parallelism exploitable at each stage:

- Parallel algorithm (A)
- High-level language (L)
- Efficient object code (O)
- Target machine code (M)

The degree of parallelism refers to the number of independent operations that can be performed simultaneously. In order to promote parallel processing in machine hardware, an intelligent compiler is needed to regenerate the parallelism through vectorization as shown in figure 4a.

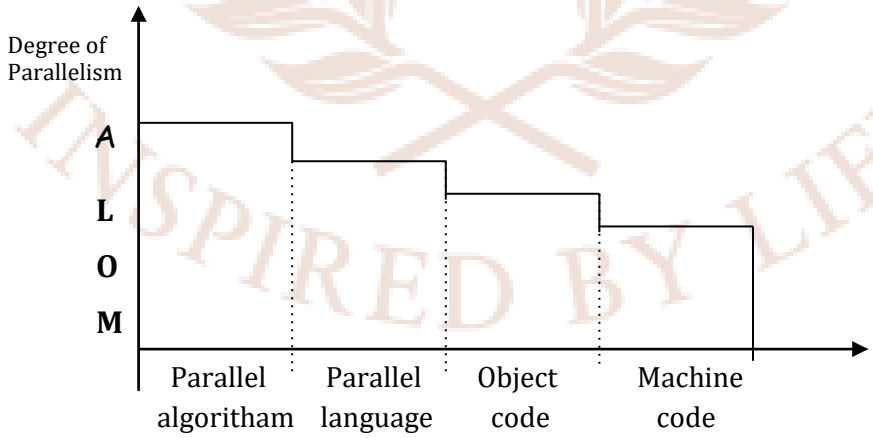


Figure 4(a): The ideal case of using parallel algorithm

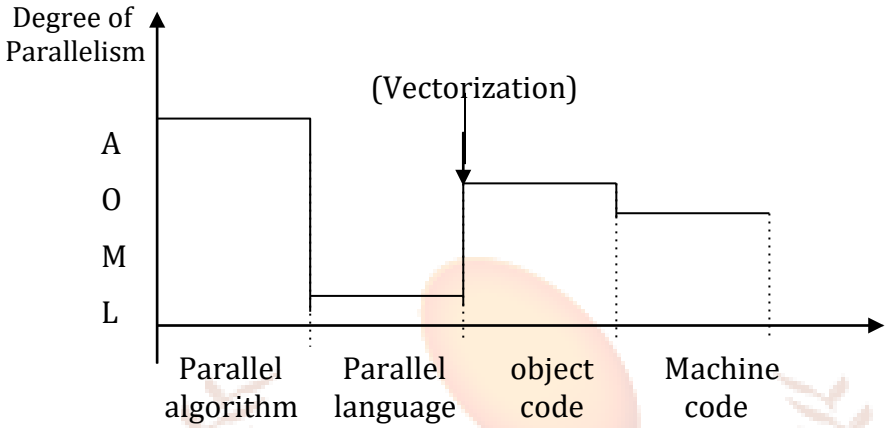


Figure 4(b): The case of using vectorizing compiler and sequential language

The process to replace a block of sequential code with vector instructions is called vectorization. The system software, which does this regeneration of parallelism, is called a vectorizing compiler(refer to figure 4b).

4.2 Multiple Vector Task Dispatching

A parallel task-scheduling model is presented for multi-pipeline vector processors. This model can be applied to explore maximum concurrency in vector supercomputers. The functional block diagram of a modern multiple-pipeline vector computer is shown in figure

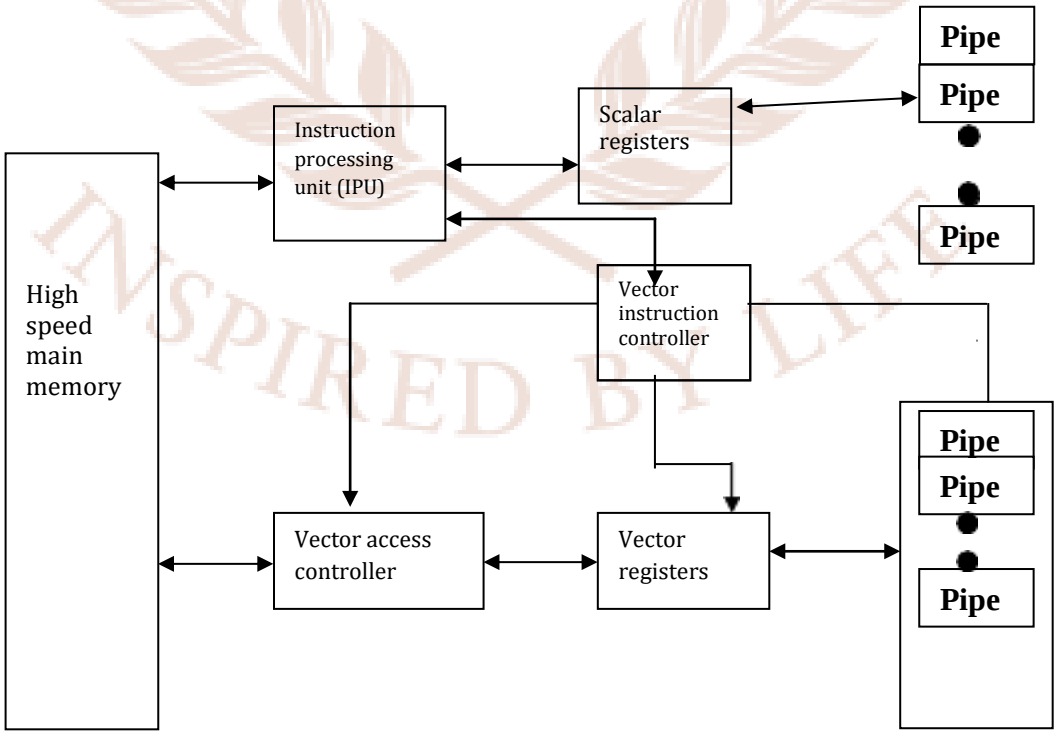


Figure 5: The architecture of a typical vector processor with multiple functional pipes

The main memory is often interleaved to minimize the access time of vector operands. Instructions and data may appear in either vector or scalar formats. *The instruction-processing unit* (IPU) fetches and decodes scalar and vector instructions. All scalar instructions are dispatched to the *scalar processor* for execution. The scalar processor itself contains multiple scalar pipelines. A *task* system contains a set of vector instructions (tasks) with a precedence relation determined only by data dependencies. A long vector task can be partitioned into many sub vectors, to be processed by several pipelines concurrently. An increase in system overhead may be incurred with vector segmentations.

5. SUPER SCALAR PROCESSES

A CISC and RISC can be improved with the superscalar or vector architecture. In a superscalar processor, multiple instruction pipelines are used. That is multiple instructions are issued per cycle and multiple results are generated per cycle.

5.1 The Emergence and Spread of Superscalar Processors

The path to widespread use of superscalar instruction issues in main-line processors took quite some time. As is usual in technology-related developments, superscalar processors *emerged in three consecutive phases*. First, the *idea* was conceived then a *few architecture proposals and prototype machines appeared,* and finally, in the last phase, the commercial *products* reached the market.

The idea of the superscalar issue was first formulated as early as 1970. It was later reformulated more precisely in the 1980s. Superscalar RISC processors emerged according to two different approaches. Some appeared as the result of converting an existing (scalar) RISC line into a superscalar one. Examples of this are the Intel 960, MC 88000, HP PA (Precision almost identical to the Architecture), and RISC lines. The other major approach was to conceive a new architecture and implement it from the very beginning as a superscalar line.

Owing to their higher complexity, superscalar CISC processors appeared on the market after a considerable delay. Higher than RISC complexity is caused by two reasons. First, in contrast with RISCs, superscalar CISC processors have to decode multiple variable-length instructions. Second, it is more demanding to implement CISC-type memory architecture than a simple RISC-type load/store architecture. The Pentium and the MC 68060 are examples of the first superscalar CISC machines, which have been available since 1993. Both were the result of converting existing CISC lines into superscalar ones.

All of these superscalar CISC processors have a low issue rate of around 2 due to the additional complexity of superscalar CISC processors mentioned above. We note that some CISC processors, for instance, the Pentium, the Nx586, and the K5, are implemented using a

superscalar RISC core. In the processors, the RISC core typically has an issue rate of 4, which is equivalent to a CISC rate of about 2. Clearly, the market includes different classes of superscalar processors with varying application fields, performance levels, and architectures. The Intel 960 and the Am 29000 superscalar processors are typically intended for the high-performance desktop and workstation market. The PowerPC 602 and PowerPC 603 are exceptions, being low-cost, low power models.

5.2 Specific Tasks of Superscalar Processing

Superscalar processing can be broken down into a number of specific tasks, which, we will review based on figure 6.

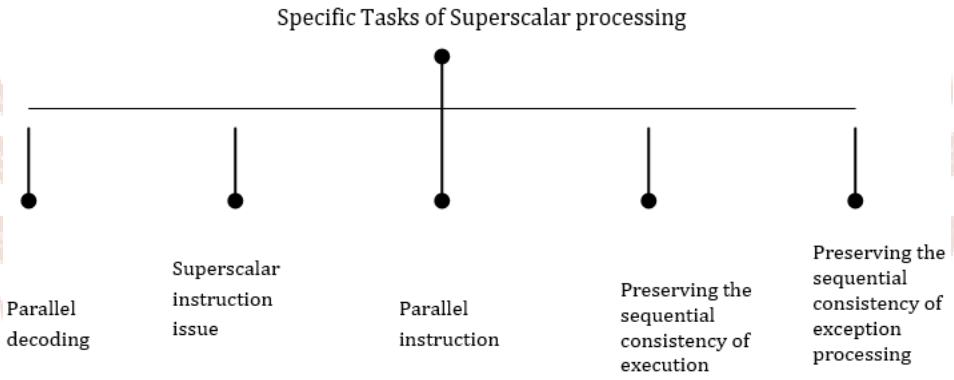


Figure 6: Specific Tasks of Superscalar Processors

Since super scalar processors have to issue multiple instructions per cycle, the first task necessarily is parallel decoding. Clearly, decoding in superscalar processors is a considerably more complex task than in the case of scalar processors and becomes even more sophisticated as the issue rate increases. Higher issue rates, however, can unduly lengthen the decoding cycle or can give rise to multiple decoding cycles unless decoding is enhanced. An increasingly common method of enhancement is pre-decoding. This is a partial, decoding performed in advance of common decoding, while instructions are loaded, into the instruction cache.

Therefore, in order to achieve higher performance, superscalar processors have introduced *intricate instruction issue policies*, involving advanced techniques such *as shelving, register*

renaming, and speculative branch processing. As a consequence, the instruction issue policy used becomes crucial for achieving higher, processor performance.

Self-Assessment Questions – 2

4. A_____processor executes vector instructions on arrays ofdata.
5. One obvious advantage of vector processing over scalar processing is the elimination of the overhead caused by_____mechanism.
6. All scalar instructions are dispatched to the *vector processor* for execution. (True/False).
7. The Pentium and the MC 68060 are examples of the first superscalar CISC machines. (True/False).

6. SUPERSCALAR INSTRUCTION ISSUE

In this section, you will be introduced to the issues concerned to the superscalar instructions.

6.1 The design space

Superscalar instruction issue is the most sensitive task of superscalar operation. It can favorably be discussed within the framework of its design space. As figure 14.7 illustrates, the superscalar instruction issue comprises two major aspects, issue policy, and issue rate.

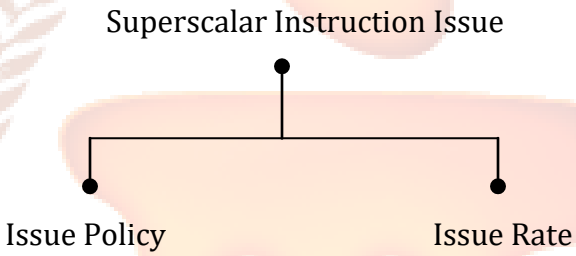


Figure 7: Design space of Superscalar Instruction Issue

The issue policy specifies how dependencies are handled during the issue process. The issuing rate, on the other hand, specifies the maximum number of instructions a superscalar processor is able to issue in each cycle.

6.2 Issue policies

The design space of the issue policy is considerably complex. It consists of four major aspects (refer to figure 8).

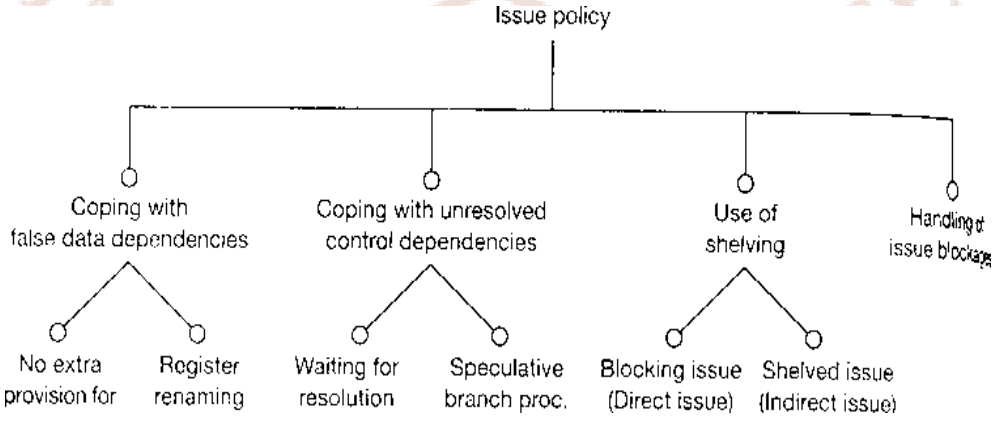


Figure 8: issue policy

The first two specify how false data and unresolved control dependencies are coped with during instruction issues. In both cases, the design options are either to avoid them during instruction issues by using register renaming and speculative branch processing, respectively, or not.

The third aspect determines whether issue blockages will be drastically reduced by the technique of shelving and the final aspect specifies how to deal with issue blockages. In the following, we provide a brief overview of the design aspects mentioned.

Within the design space of issue policy, the first aspect determines how, the processor copes with false data dependencies, which occur either between the instruction to be issued and those in execution or among the instructions to be issued. Specifically, we are concerned with the decision of whether WAR (write after read) and WAW (write after write) dependencies occurring between register references will be eliminated or not. We emphasize that this aspect is confined to register data dependencies and does not cover possible false data dependencies between memory data. The rationale, therefore, is that false data dependencies between memory data are much less frequent than those between register data, especially in RISC architectures. False data dependencies between register data may be removed by register renaming. *The principle of register renaming* is quite simple. If a WAW or WAR dependency is encountered, the destination register causing the dependency is renamed. This means that the result of the instruction causing a WAW or WAR dependency is written not into the specified register, but into a dynamically allocated 'spare register' instead.

The second aspect of issue policies determines how *unresolved control dependenci*es are dealt with. Obviously, a conditional control transfer instruction causes problems if the condition has not yet been produced by the generating instruction by the time the condition should be evaluated. Issue policies handle this in two different ways. Either they wait until the referenced condition becomes available, or they employ speculative execution. With *speculative execution of control transfer instruction*, *or speculative branch processing* for

short, a guess is made as to the outcome of each unresolved conditional transfer instruction. The instruction issue is then resumed accordingly.

The third aspect of issue policy concerns the *use of shelving*, an efficient technique to avoid issue blockages. Simply speaking, the alternatives are either to allow dependent instructions to cause issue blockages *(blocking issues*) or to avoid than through shelved issues*. The blocking issue* (also called direct issue) is a strict approach related to dependencies. In this scheme decoded instructions to be issued are checked for dependencies. When dependencies occur, the instruction issue is blocked. *Shelving* (also called the shelved issue, indirect issue) decouples instruction issues and dependency checking. This technique presumes that special instruction buffers often referred to as reservation stations are provided in front of the execution units (EU). With shelving, instructions are issued first to the shelving buffers with essentially no checks for dependencies.

When shelving is used, no dependency checks are performed between the instructions in the issue window and those in execution. This kind of dependency checking is delayed to a later step of processing called dispatching. Nevertheless, if renaming is employed, an inter-instruction dependency check is needed for the instructions in the issue window. During *dispatching* the instruction held in the shelving buffers are checked for dependencies and dependency-free instructions are forwarded to available EUs.

There are two different terms, instruction issue, and dispatch, to express different actions. The term '*issue*' is used with two different interpretations. Without shelving, that is, when the blocking issue is used, *'issue*' refers to, the action of disseminating decoded independent instructions to the EUs. When shelving is used, *'issue'* designates the dissemination of decoded instructions to the shelving buffers. The term '*dispatch*', on the other hand, is only used in the case of shelving. Here dispatch designates the dissemination of dependency-free instructions from, the shelving buffers to the EUs. The last aspect of the design space of instruction issue declares how to issue blockages are handled. Instruction issues may be blocked in both cases, with or without shelving. As long as shelving is not used, any dependencies encountered in the issue window immediately block the issue of instructions.

In contrast, when shelving is used, issue blockages due to data and unresolved control dependencies are avoided.

6.3 Instruction Issue Policies of Scalar Processors

While discussing instruction issue policies in the case of scalar processors, we have to take into account the three basic issue aspects, that is, whether renaming, speculative execution, and shelving are employed or not.

Scalar processors use predominantly two issue policies. We refer to them as the traditional scalar issue and its enhancement *with speculative execution*. In addition, there are two further policies of historical importance, designated as the traditional scalar issue with shelving and the *traditional scalar issue with shelving and renaming*.

6.4 Instruction Issue Policies of Superscalar Processors

When we expand our discussion of instruction policies to superscalar processors, we have to take into account, beyond the three basic issue aspects considered above. There are 12 feasible issue policies. Of these superscalar processors employ mainly five.

They are designated, the *straightforward superscalar issue with and without issue alignment, the straightforward superscalar issue with shelving*, the *straightforward superscalar issue with renaming*, and the *advanced superscalar issue*. The simplest, most often used policy is the *straightforward aligned superscalar issue*. This policy does not provide for renaming, uses an aligned blocking issue, and employs speculative execution. According to this policy, instruction issue is blocked for data and resource dependencies, whereas control dependencies are met with speculative branch processing. Another group of processors makes use of the *straightforward superscalar issue*, in a more advanced form, with an *alignment-free issue.* This additional feature contributes to increased performance.

The next issue policy, *the straightforward superscalar issue policy*, with shelving, does not employ renaming, handles unresolved control dependencies with speculative execution, and introduces shelving. As its name implies, the straightforward superscalar issue policy with

renaming, the fourth scheme offers renaming and speculative branch processing but sticks at the blocking issue. It actually has a scalar issue with some enhancements, which appears only in certain situations as a superscalar processor.

The final policy commonly used is the *advanced superscalar issue policy*. At present this policy is the most relevant one. It employs register renaming, speculative branch processing, and shelving. According to this policy, false register data dependencies are eliminated by register renaming, speculative execution is used to cope with unresolved control dependencies and shelving removes issue blockages due to dependencies, provided that enough free shelving buffer entries and wide enough data paths are available. Obviously, this issue policy is the most advance in the framework of the design space considered.

Register renaming is a standard technique for removing false data dependencies, that is, WAR and WAW dependencies, among register data.

Register renaming presumes the three operand instruction format. It may be implemented either statically or dynamically. In the case of static implementation, register renaming is carried out during compilation. Register renaming is implemented dynamically, renaming takes place during execution.

7. SUMMARY

In this unit, we discussed about parallel organization. We also discussed on clusters which are a collection of independent uniprocessors. Then we discussed CISC and RISC processors. We discussed these two systems with respect to their characteristics and a comparison was done referring to instruction set complexity and architectural distinctions. Then we have concentrated on a vector processor that executes vector instructions on arrays of data. We have seen the vector processing characteristics that explain why most vector processors have a pipeline structure. A parallel task-scheduling model is presented for multi-pipeline vector processors. This model can be applied to explore maximum concurrency in vector supercomputers. Finally, we have introduced the superscalar processors. Superscalar processing can be broken down into a number of specific tasks that we have seen in this unit. We have also touched upon the issue policies for both Scalar processors and superscalar processors.

8. TERMINAL QUESTIONS

1. What is parallel organization? Explain.
2. What is UMA? Explain.
3. What are the characteristics of RISC and CISC architectures?
4. What are the characteristics of Vector Processing
5. Write short note on super Scalar Processors.
6. Discuss the issues of super Scalar instruction.

9. ANSWERS

Self-Assessment Questions

1. CISC
2. Instructions
3. Control units
4. Hardwired
5. Vector
6. False
7. True

Terminal Questions

1. In a parallel organization, multiple processing units cooperate to execute applications. A parallel processing organization looks for a grosser level of parallelism, one that enables work to be done in parallel, and cooperatively, by multiple processors. Refer section 2.
2. UMA is a shared memory architecture used in parallel computers. Refer section 2.
3. Major characteristics of CISC architecture are: A large number of instructions - typically from 100 to 250 instructions, some instructions that perform a specialized task and are used infrequently. Refer sub- section 3.2.
4. A vector operand contains an ordered set of n elements, where n is called the length of the vector. Refer sub-section 4.1.
5. In a superscalar processor, multiple instruction pipelines are used. That is multiple instructions are issued per cycle and multiple results are generated per cycle. Refer section 5.
6. Superscalar instruction issue is the most sensitive task of super scalar operation. It can favorably be discussed within the framework of its design space. Refer section 6.

Acknowledgments, References and Suggested Readings

- Jordan H. Computer System Design & Architecture. PHI
- William Stalling," Computer Organization and Architecture", Prentice Hall, Person Education Asia
- John P. Hayes, "Computer Architecture and Organization", McGraw Hill,
- Tannenbaum, "Computer Organization", PHI
- V. Carl Hamacher and Zaky, "Computer Organization", McGraw Hill.
- Bartee C Thomas. "Computer Architecture and Logic Design", Tata McGraw Hill.
- Mano Moris, "Computer System Architecture", Prentice Hall of India, Second Edition
- Wang H Kai. Advanced Computer, Architecture Parallelism, Scalability and Programmability. PHI
- Patterson A David, Hennessy. Computer Organization and Design: The Hardware/Software Interface.
- Godse D.A., Godse A.P. Computer Organization

- Sajjan G. Shiva; Computer Design and Architecture. Marcel Dekker
- Nicholas P. Carter; Schaum's outline of computer Architecture; Mc. Graw-Hill Professional

E-references

- http://www.csi.ucd.ie/staff/jcarthy/home/alp/alp-05.pdf
- http://www.windowsnetworking.com/articles_tutorials/direct-memory access.html
- http://npeducation.blogspot.in/2011/01/8255-programable-peripheral-interface.html
- http://techtips.salon.com/information-computer-peripherals-13068.html
- http://www.crm.mb.ca/guide/components/scanner.html
- http://publib.boulder.ibm.com/infocenter/zos/basics/topic/com.ibm.zos.zconcepts/zconcepts_75.html/
- http://www.ibm.com/search/csass/search?sn=mh&q=multiprocessing%20system&lang=en&cc=us&/
- www.scribd.com/doc/53304318/95/INPUT-OUTPUT-PROCESSOR-IOP
- http://euler.mat.uson.mx/~havillam/ca/CS323/0708.cs-323003.html
- http://doit.ort.org/course/cont_frame.htm