NAME :	V.POORNA CHANDRA
REGNO :	18BCE0209
COURSE :	CSE 2001
FACULTY :	BHULAKSHMI BONTHU
SLOT :	G2
DUE ON :	21-OCT-2019

Question:

Analyse any four different processors architecture in terms of
Pipelining
Non pipelining
ILP
Branch prediction

Pipelining:

Pipelining is a process of accumulating instruction from a processor through a pipeline.It allows storing and executing instructions in an orderly process.
It is also known as pipeline processing.

In pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and combinational circuit performs operations on it.The output of the combinational circuit is applied to the input register of the next segment.

Non Pipelining:

All the actions (fetching, decoding, executing of instructions and writing the results into the memory) are grouped into a single step.
It has a low throughput.
Only one instruction is executed per unit time and execution process requires more number of cycles.

The CPU scheduler in the case of non-pipelining system merely chooses from the pool of waiting work when an execution unit gives a signal that it is free.

It is not dependent on CPU scheduler.

Instruction Level Parallelism:

Pipelining can overlap the execution of instructions when they are independent of one another. This potential overlap among instructions is called instruction-level parallelism (ILP) since the instructions can be evaluated in parallel.

To obtain substantial performance enhancements, we must exploit ILP across multiple basic blocks.
The simplest and most common way to increase the amount of parallelism available among instructions is to exploit parallelism among iterations of a loop. This type of parallelism is often called loop-level parallelism.

Branch Prediction:

Branch prediction is a technique used in CPU design that attempts to guess the outcome of a conditional operation and prepare for the most likely result. A digital circuit that performs this operation is known as a branch predictor. It is an important component of modern CPU architectures, such as the x86.

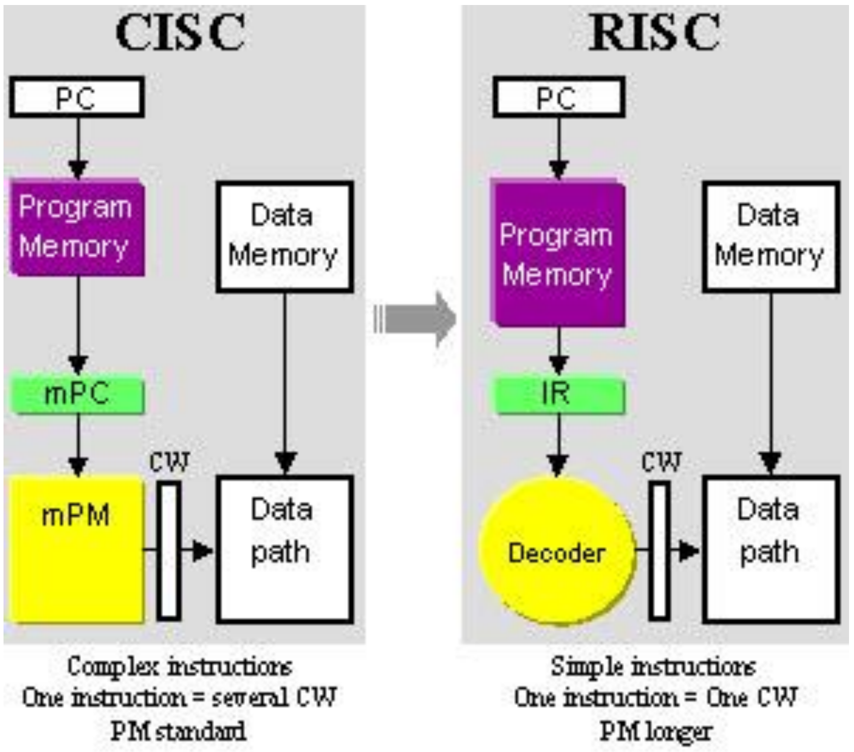
Different Architectures:

COMPLEX INSTRUCTION SET COMPUTER(CISC):

Previously in earlier days, codes were written in Machine or Assembly code. Writing codes in machine language was a very difficult task. To make working easier computers supported instructions. A single instruction could fetch more than one operand and do more than one operation on these operands. This made it easier for programmers, as they can write lesser codes. Another advantage is the memory cost. CISC reduces the memory requirements.

REDUCED INSTRUCTION SET(RISC):

Complex instructions in CISC take many processor cycles to execute. For a pipelined processor, overall speed depends upon the slowest operation that is being performed. Also complex instructions will slow down the execution of even simple instructions. One trait of RISC is its load/store architecture, where memory is accessed through specific instructions rather than as a part of most instructions. There are various designs of RISC which includes, ARC, Alpha, Am29000, ARM, Atmel AVR. Also ARM architecture processors are used in smartphones and tablets. RISC processors are also used in supercomputers such as Summit, which is the world’s fastest supercomputer.



As compiler technology has upgraded, programmers are now using high level languages. It is the task of compiler to translate high level code to assembly language code. Compilers use a combination of simple instructions to achieve this. However, it was found that breaking complex operations into simple operations was much more efficient. So most of the complex instructions offered were not being used my compiler generated programs. Processor designers started to reduce size and complexity of instruction sets. There are 2 advantages of doing this,

- Simpler instructions will speed up pipeline and provide performance boost.

- Simple instructions require simple and less hardware, which reduces cost.

5 STAGE RISC PIPELINE:

1) Instruction Fetch:

- Instruction Cache on these machines has a latency of one cycle, which means that if the instruction was in the cache, it will be ready on the next clock cycle. During the Instruction Fetch stage, a 32-bit instruction will be fetched from the cache.

- Program Counter is a register which holds the address of current instructions. It feeds into the PC predictor, which will eventually sends Program Counter (PC) to Instruction Cache to read the current instruction. At same time, the PC predictor predicts the address of the next instruction by incrementing the PC by 4.

2) Instruction Decode:

- First RISC machines had no computer code. Once it's fetched from instruction cache, the instruction bits are going to be shifted down the pipeline, in order that easy combinatory logic in every pipeline stage may turn out the management signals for the data path directly from the instruction bits As a result, little coding is completed within the stage historically referred to as the decipher stage. A consequence of this lack of coding meant but that a lot of instruction bits had to be used specifying what the instruction ought to do which leaves fewer bits for things like register indices.The decipher stage over up with quite an heap of hardware: unit of measurement had the likelihood of branching if 2 registers were equal, therefore a 32-bit-wide AND tree ran nonparallel once the register file browse, creating a really long crucial path through this stage. Also, the branch target computation usually needed a sixteen bit add and a fourteen bit incrementer. resolution the branch within the decipher stage created it doable to possess simply a single-cycle branch mispredict penalty. Since branches were fairly often taken, it had been important to stay this penalty low.

3) Execute:

• The Execute stage is wherever the particular computation happens. Generally this stage consists of associate Arithmetic and Logic Unit, and additionally a

- The Arithmetic and Logic Unit is liable for activity Boolean operations
  and additionally for activity whole number addition and subtraction. Besides the result, the ALU generally provides standing bits like whether or not or not the result was zero, or if associate overflow occurred.
- The bit shifter is liable for shift and rotations.

4) Memory Access:

• If memory needs to be accessed, it will be done in this stage. touch shifter. it should additionally embrace a multiple cycle number and divider.

• In this stage, single cycle latency instructions will simply forward their results to the next stage. This forwarding ensures that both one and two cycle instructions always write their results in the same stage of the pipeline so that only one write port to the register file could be used, and will always be available.

5) Write back:

• In this stage, both single cycle and two cycle instructions write their results into register file.

Branch Prediction:

- A branch predictor is a digital circuit that tries to guess which way a branch (e.g. an if–then–else structure) will go before this is known definitively.
- The early implementations of SPARC and MIPS (two of the first commercial RISC architectures) used single-direction static branch prediction: they always predict that a conditional jump will not be taken, so they always fetch the next sequential instruction. Only when the branch or jump is evaluated and found to be taken, does the instruction pointer get set to a non-sequential address.
- Both CPUs evaluate branches in the decode stage and have a single cycle instruction fetch. As a result, the branch target recurrence is two cycles long, and the machine always fetches the instruction immediately after any taken branch. Both architectures define branch delay slots in order to utilize these fetched instructions.
- A more advanced form of static prediction presumes that backward branches will be taken and that forward branches will not. A backward branch is one that has a target address that is lower than its own address. This technique can help with prediction accuracy of loops, which are usually backward-pointing branches, and are taken more often than not taken.
- Some processors allow branch prediction hints to be inserted into the code to tell whether the static prediction should be taken or not taken.

SUPERSCALAR PROCESSORS:

Instruction level parallelism:

In Superscalar execution, multiple execution units are used to execute multiple instructions in parallel, which is known as Instruction level parallelism.

The simplest processors are scalar processors. every instruction dead by a scalar processor generally manipulates one or 2 information things at a time. in contrast, every instruction dead by a vector processor operates at the same time on several information things. Associate in Nursing analogy is that the distinction between scalar and vector arithmetic. A superscalar processor could be a mixture of the 2. every instruction processes one information item, however there are multiple execution units among every process unit therefore multiple directions is processing separate information things at the same time.

Superscalar central processing unit style emphasizes rising the instruction dispatcher accuracy, and permitting it to stay the multiple execution units in use the least bit times. This has become more and more vital because the variety of units has multiplied. whereas early superscalar CPUs would have 2 ALUs and one FPU, a later style like the PowerPC 970 includes four ALUs, two FPUs, and 2 SIMD units. If the dispatcher is ineffective at keeping all of those units fed with directions, the performance of the system are no higher than that of an easier, cheaper style.

A superscalar processor typically sustains Associate in Nursing execution rate in far more

than one instruction per machine cycle. however simply process multiple directions at the same time doesn't build Associate in Nursing design superscalar, since pipelined, digital computer or multi-core architectures conjointly succeed that, however with totally different strategies.

In a superscalar central processing unit the dispatcher reads directions from memory and decides which of them is run in parallel, dispatching every to 1 of the many execution units contained within one central processing unit. Therefore, a superscalar processor is visualized having multiple parallel pipelines, every of that is process directions at the same time from one instruction thread.

A superscalar processor typically sustains Associate in Nursing execution rate in far more

than one instruction per machine cycle. however simply process multiple directions at the same time doesn't build Associate in Nursing design superscalar, since pipelined, digital computer or multi-core architectures conjointly succeed that, however with totally different strategies.

In a superscalar central processing unit the dispatcher reads directions from memory and decides which of them is run in parallel, dispatching every to 1 of the many execution units contained within one central processing unit. Therefore, a superscalar processor is visualized having multiple parallel pipelines, every of that is process directions at the same time from one instruction thread.

Challenges in ILP:

1) The Hardware Model:

An ideal processor is one where all artificial constraints on ILP are removed. The only limits on ILP in such a processor are those imposed by the actual data flows either through registers or memory.

The assumptions made for an ideal or perfect processor are as follows:

1.**Register renaming:** There are a large number of virtual registers available and hence all WAW and WAR hazards are avoided and an unbounded number of instructions can begin to execute simultaneously.

2.**Branch prediction:** Branch prediction is perfect. All conditional branches are predicted exactly.

3.**Jump prediction:** All jumps are perfectly predicted. When combined with perfect branch prediction, this is equivalent to having a processor with perfect speculation and an unbounded buffer of instructions available for execution.

4.**Memory-address alias analysis:** All memory addresses are known exactly and a load can be moved before a store provided that the addresses are not identical.

2) **Limitations on the Window Size and Maximum Issue Count**:

A dynamic processor might be able to more closely match the amount of parallelism uncovered by our ideal processor. Consider what the perfect processor must do:

1.Look arbitrarily far ahead to find a set of instructions to issue, predicting all branches perfectly.

2.Rename all register uses to avoid WAR and WAW hazards.

3.Determine whether there are any data dependencies among the instructions in the issue packet; if so, rename accordingly.

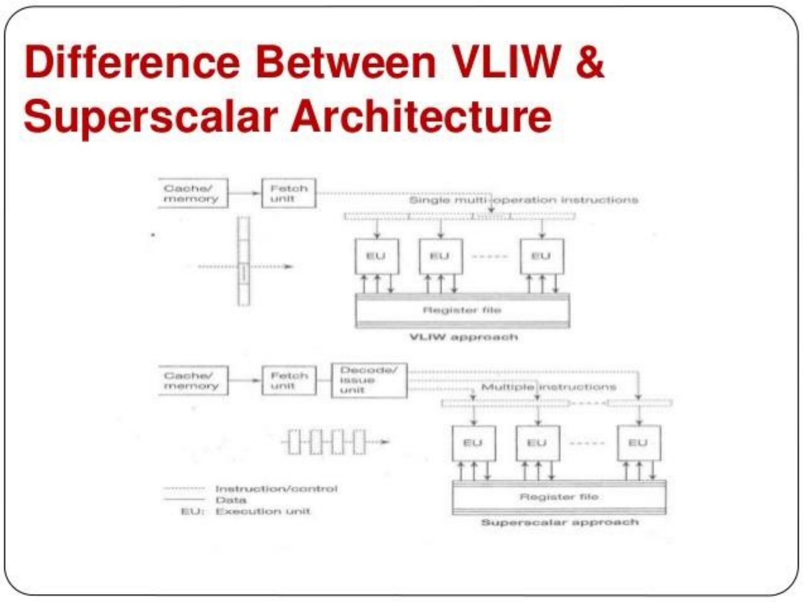
4.Determine if any memory dependences exist among the issuing instructions and handle them appropriately.

5.Provide enough replicated functional units to allow all the ready instructions to issue.

VLIW ARCHITECTURE:

This architecture consists of multiple ALUs in parallel.

These architectures are designed to make use of applications of Instruction Level Parallelism. Programmer break their code such that each ALU can be loaded in parallel. Operation to be done on each ALU forms an instruction word. It is up to programmer to take care of partitioning of application across different ALUs. Programmer should take care that there is no independency between which are part of instruction word.



Difference Between VLIW & Superscalar Architecture

Superscalar Architectures are similar to VLIW architecture in a way, as both of them keep multiple ALUs. However, superscalar processors employ dynamic scheduling of instructions. They have separate hardware to take care of the independency.

Instruction level processing:

Very-Long Instruction Word (VLIW) architectures are an appropriate various for exploiting instruction-level similarity (ILP) in programs, that is, for execution over one basic instruction at a time. These processors contain multiple useful units and fetch from the instruction cache
, a Very-Long Instruction Word containing many primitive directions, and dispatch the complete VLIW for parallel execution. These capabilities are exploited by compilers that generate code that has sorted along freelance primitive directions feasible in parallel. The processors have comparatively straightforward management logic as a result of neither do they perform any dynamic programming nor rearrangement of operations .

VLIW has been represented as a natural successor to reduced instruction set computing, as a result of it moves quality from the hardware to the compiler, permitting less complicated, quicker processors. the target of VLIW is to eliminate the sophisticated instruction