19. The branch type instructions in program cause [WBUT 2016]

a) data hazard b) structural hazard
c) control hazard d) none of these

Answer: (c)

20. The instruction execution flow in the pipeline processor is represented by [WBUT 2016]

a) reservation table b) data-flow diagram
c) time-space diagram d) flow chart

Answer: (a)

21. A no. of pipelines that are working in parallel are used in [WBUT 2016]

a) pipelining b) super-pipelining
c) superscalar d) VLIW processor

Answer: (b)

22. What will be the speed up for 4 stage linear pipeline for no. of instructions = 64? [WBUT 2017]

a) 4.5 b) 6.5 c) 7.1 d) none of these.

Answer: (d)

Short Answer Type Questions

1. What is understood by dynamic pipeline? [WBUT 2006]

Answer:

It is a pipeline with dynamic scheduling. Dynamic scheduling is one method for improving performance in a multiple instruction issue processor. When applied to a super scalar processor, dynamic scheduling will boost up performance in the face of data hazards and also allows the processor to potentially overcome the issue restrictions. Dynamic pipeline must be multifunctional. In the dynamic pipeline different reservation table are used for different function.

2. What is the function of reservation table in pipeline architecture system? [WBUT 2006]

Answer:

The functions of reservation table are as follows:

- Reservation table displays the time-space flow of data through the pipeline for one function evaluation.
- Different functions may follow different paths.
- The same table may represent a number of pipeline configurations.
- The number of columns in a reservation table is called the evaluation time.

3. What is parallel algorithm? [WBUT 2006]

Answer:

Most of today's algorithms are sequential, that is, they specify a sequence of steps in which each step consists of a single operation. These algorithms are well suited to today's

computers, which basically perform operations in a sequential fashion. Although the speed at which sequential computers operate has been improving at an exponential rate for many years, the improvement is now coming at greater and greater cost. As a consequence, researchers have sought more cost-effective improvements by building "parallel" computers – computers that perform multiple operations in a single step. In order to solve a problem efficiently on a parallel machine, it is usually necessary to design an algorithm that specifies multiple operations on each step, i.e., a parallel algorithm. An efficient way of information processing which efficiently handles concurrent events in processing environment. The algorithm efficiently allocates limited hardware-software resources to multiple programs in large computational problems or parallel processing conducted among procedures or tasks within the same program.

4. Give an example of concurrent processes in computation? **[WBUT 2006]**

Answer:

There is a prime number generator pipeline as shown in figure (a). Integer numbers are generated and successively tested for divisibility by previously generated primes in a linear pipeline of primes. The circle numbers represent those being generated. A number enters the pipeline from the left end and is eliminated if it is divisible by the prime number tested at a pipeline stage. All the numbers being forwarded to the right of a pipeline stage are those indivisible by all the prime numbers tested on the left of that stage.

In the next figure (b) we shows that the multiplication of a list of numbers (10, 7, -2, 3, 4, -11, -3) using a divide-and-conquer approach. The numbers are represented as leaves of a tree. The problem can be subdivided into subproblems of multiplying two sublists, each of which is concurrently evaluated and the results multiplied at the upper node.

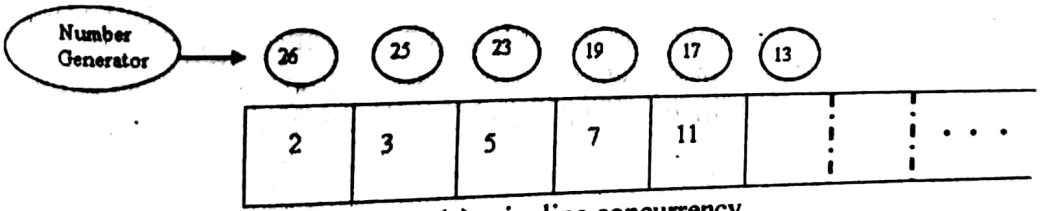


Figure (a): pipeline concurrency

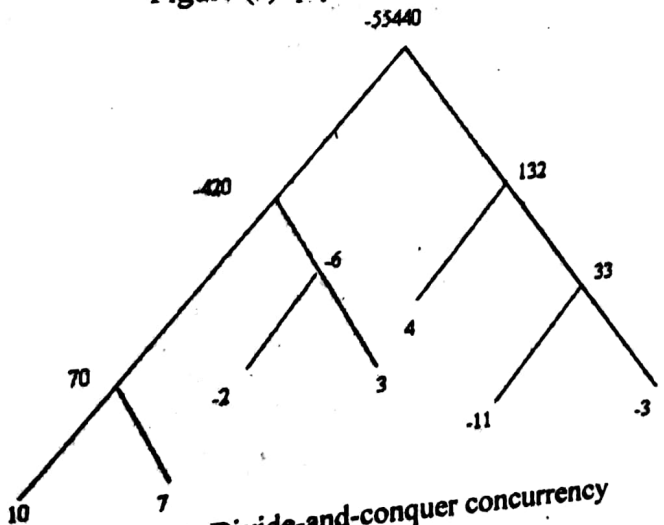


Figure (b): Divide-and-conquer concurrency

5. Define Speed-up. Deduce that the maximum speed-up in a k-stage pipeline processor is k. Is this maximum speed-up always achievable? Explain.

OR, **[WBUT 2006]**

Show that the maximum speed-up of a pipeline is equal to its number of stages. **[WBUT 2014]**

Answer: ***Refer to Question No. 9(c) of Long Answer Type Questions.***

6. Compare superscalar, superpipeline and superscalar superpipelined architecture. **[WBUT 2007, 2012]**

Answer:

Superscalar and super-pipelined processors utilize parallelism to achieve peak performance that can be several times higher than that of conventional scalar processors. Superscalar machines can issue several instructions per cycle. A system was developed and used to measure instruction-level parallelism for a series of benchmarks. The average degree of super pipelining metric is introduced. Our simulations suggest that this metric is already high for many machines. These machines already exploit all of the instruction-level parallelism available in many non-numeric applications, even without parallel instruction issue or higher degrees of pipelining.

Superscalar

Superscalar processing has multiple functional units are kept busy by multiple instructions. As execution pipelines have approached the limits of speed, parallel execution has been required to improve performance. Super-pipelined machines can issue only one instruction per cycle, but they have cycle times shorter than the latency of any functional unit. In some cases superscalar machines still employ a single fetch-decode-dispatch pipe that drives all of the units. For example, the Ultra SPARC splits execution after the third stage of a unified pipeline. However, it is becoming more common to have multiple fetch-decode-dispatch pipes feeding the functional units. Superscalar operation is limited by the number of independent operations that can be extracted from an instruction stream.

Super-pipeline

Given a pipeline stage time T, it may be possible to execute at a higher rate by starting operations at intervals of T/n. This can be accomplished in two ways:

- Further divide each of the pipeline stages into n substages.
- Provide n pipelines that are overlapped.

The first approach requires faster logic and the ability to subdivide the stages into segments with uniform latency. The second approach could be viewed in a sense as staggered superscalar operation, and has associated with it all of the same requirements except that instructions and data can be fetched with a slight offset in time.

Super-pipelining is limited by the speed of logic, and the frequency of unpredictable branches. Stage time cannot productively grow shorter than the inter stage latch time, and so this is a limit for the number of stages. The MIPS R4000 is sometimes called a super-pipelined machine. The benefit of such extensive pipelining is really only gained for very regular applications such as graphics.

Superscalar-Super-pipeline

We may also combine superscalar operation with super-pipelining and the result is potentially the product of the speedup factors. However, it is even more difficult to interlock between parallel pipes that are divided into many stages. Also, the memory subsystem must be able to sustain a level of instruction throughput corresponding to the total throughput of the multiple pipelines -- stretching the processor/memory performance gap even more. Of course, with so many pipes and so many stages, branch penalties become huge, and branch prediction becomes a serious bottleneck.

But the real problem may be in finding the parallelism required to keep all of the pipes and stages busy between branches. Consider that a machine with 12 pipelines of 20 stages must always have access to a window of 240 instructions that are scheduled so as to avoid all hazards, and that the average of 40 branches that would be present in a block of that size are all correctly predicted sufficiently in advance to avoid stalling in the prefetch unit.

7. What are the different factors that can affect the performance of a pipelined system? Differentiate between WAR and RAW with a suitable example.

[WBUT 2007, 2010]

OR,

What are deferent factor affect the pipeline performance? Differentiate between WAR and RAW?

[WBUT 2017]

Answer:

Pipelining achieves a reduction of the average execution time per instruction. In the sense that pipeline can perform more instructions per clock cycle. This can be viewed in two ways:

- Decreasing the CPI. Typical way in which people view the performance increase
- Decreasing the cycle time (i.e., increasing the clock rate).

Pipelining increases the CPU instruction throughput. Pipelining does not decrease the execution time of an individual instruction. It increases the execution time due to overhead (clock skew and pipeline register delay) in the control of the pipeline.

Data hazards occur when data is modified. Ignoring potential data hazards can result in race conditions. There are two situations a data hazard can occur in:

WAR hazards	RAW hazards
1. j tries to write a destination before it is read by i , so i incorrectly gets the new value.	1. j tries to read a source before i writes it, so j incorrectly gets the old value.
2. WAR hazard is eliminated by register renaming of all the destination registers including those with a pending read or write for an earlier instruction.	2. RAW hazards are avoided by executing an instruction only when its operands are available.
3. WAR hazard if $D(i) \cap R(j) \neq \emptyset$	3. RAW hazard if $R(i) \cap D(j) \neq \emptyset$
4. For example: i1. R4 ← R1 + R3 i2. R3 ← R1 + R2	4. For example: i1. R2 ← R1 + R3 i2. R4 ←R2 + R3

WAR hazards	RAW hazards
If we are in a situation that there is a chance that i2 may be completed before i1 (i.e. with concurrent execution) we must ensure that we do not store the result of register 3 before i1 has had a chance to fetch the operands.	The first instruction is calculating a value to be saved in register 2, and the second is going to use this value to compute a result for register 4. However, in a pipeline, when we fetch the operands for the 2nd operation, the results from the first will not yet have been saved, and hence we have a data dependency.

8. What are the different parameters used in measuring CPU performance? Briefly discuss each. **[WBUT 2008]**

Answer:

To estimate the CPU performance, the measure that is generally most important is **execution time, T**. because we can write

Performance = 1 / Execution time

So, if the execution time increases the CPU performance decreases. There are three parameters to measure the performance of the CPU, i.e. speedup, efficiency and throughput.

When considering the impact of some performance improvement, the effect of the improvement is usually expressed in terms of the **speedup, *S***, taken as the ratio of the execution time without the improvement (T_{wo}) to the execution time with the improvement (T_w): $S = T_{wo} / T_w$

Speed-up as a direct percent can be represented as: $S = ((T_{wo} - T_w) / T_w) \times 100$

Efficiency, E is the ratio of Speed-up to the number of processor used.

So, $E = S / p$

Where S is the speed-up and p is the number of processor.

Throughput is the measure of number of computation over a unit time.

9. Compare superscalar, super-pipeline and VLIW techniques. **[WBUT 2008, 2011, 2012]**

OR,

Discuss about the performance of VLIW processor **[WBUT 2014]**

OR,

What is VLIW processor? How it is differ from superscalar architecture? How will you design a VLIW processor? **[WBUT 2016]**

Answer:

Superscalar

A superscalar processor executes more than one instruction during a clock cycle by simultaneously dispatching multiple instructions to redundant functional units on the processor. A superscalar CPU architecture implements a form of parallelism called instruction-level parallelism within a single processor. Superscalar processing has multiple functional units are kept busy by multiple instructions. In some cases superscalar machines still employ a single fetch-decode-dispatch pipe that drives all of the units.

Superscalar operation is limited by the number of independent operations that can be extracted from an instruction stream. It has been shown in early studies on simpler

processor models, that this is limited, mostly by branches, to a small number. The superscalar technique is traditionally associated with several identifying characteristics. These characteristics are,

- Instructions are issued from a sequential instruction stream
- CPU hardware dynamically checks for data dependencies between instructions at run time (versus software checking at compile time)
- Accepts multiple instructions per clock cycle.

Super-pipeline

Given a pipeline stage time T, it may be possible to execute at a higher rate by starting operations at intervals of T/n. This can be accomplished in two ways:

- Further divide each of the pipeline stages into n substages.
- Provide n pipelines that are overlapped.

The first approach requires faster logic and the ability to subdivide the stages into segments with uniform latency. The second approach could be viewed in a sense as staggered superscalar operation, and has associated with it all of the same requirements except that instructions and data can be fetched with a slight offset in time.

Super-pipelining is limited by the speed of logic, and the frequency of unpredictable branches. Stage time cannot productively grow shorter than the inter stage latch time, and so this is a limit for the number of stages. The MIPS R4000 is sometimes called a super-pipelined machine. The benefit of such extensive pipelining is really only gained for very regular applications such as graphics. On more irregular applications, there is little performance advantage.

VLIW

Superscalar and VLIW architectures both exhibit instruction- level parallelism, but differ in their approach. It thereby allows faster CPU throughput than would otherwise be possible at the same clock rate. Each functional unit is not a separate CPU core but an execution resource within a single CPU such as an arithmetic logic unit, a bit shifter, or a multiplier. Superscalar CPU design emphasizes improving the instruction dispatcher accuracy, and allowing it to keep the multiple functional units in use at all times.

Very Long Instruction Word (VLIW) refers to a CPU architecture designed to take advantage of instruction level parallelism. A processor that executes every instruction one after the other i.e. a non-pipelined scalar architecture may use processor resources inefficiently, potentially leading to poor performance. The performance can be improved by executing different sub-steps of sequential instructions simultaneously, or even executing multiple instructions entirely simultaneously as in superscalar architectures. The VLIW approach executes operation in parallel based on a fixed schedule determined when programs are compiled. Since determining the order of execution of operations is handled by the compiler.

10. What is meant by pipeline stall? **[WBUT 2008]**

Answer:

A pipeline operation is said to have been stalled if one unit (stage) requires more time to perform its function, thus forcing other stages to become idle. Consider, for example, the case of an instruction fetch that incurs a cache miss. Assume also that a cache miss

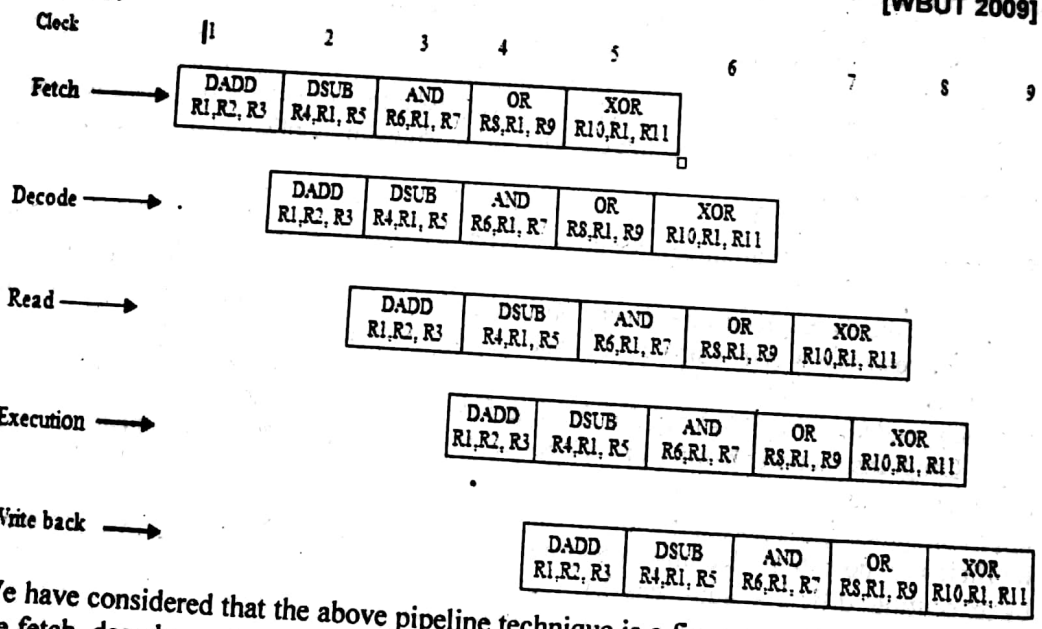
requires three extra time units. By this method we can prevent branch and structural hazards from occurring. As instructions are fetched, control logic determines whether or not a hazard could/will occur. If this is true, then the control logic inserts NOPs (No Operations) into the pipeline. Thus, before the next instruction (which will cause the hazard) is executed, the previous one will be sufficiently complete to prevent the hazard. If the number of NOPs is equal to the number of stages in the pipeline, the processor has been cleared of all instructions and can proceed free from hazards.

11. Consider the pipelined execution of these instructions:

DADD	**R1, R2, R3**
DSUB	**R4, R1, R5**
AND	**R6, R1, R7**
OR	**R8, R1, R9**
XOR	**R10, R1, R11**

Explain how the above execution may generate a data hazard and describe a way to minimize the data hazard stalls using forwarding. Modify the above example to show a case where forwarding may not work. **[WBUT 2009]**

Answer:



We have considered that the above pipeline technique is a five stage pipeline. The stages are fetch, decode, read, execution and write back. In the above figure, we also show that in which clock pulse what operation is performed.

Now, the data hazard will occur in 4th clock cycle, where both read and execution operations are performed on register R1 at the same time.

Result forwarding is a technique to minimize the stall in pipeline processing. The technique is that after execution of one instruction, if the result is transfer to the next instruction bypass the write back stage directly. i.e. without writing the result in the register pipeline transfer that result to the next instruction.

Result forwarding may not work in case of branch instruction of a pipeline execution. So, if there is a branch instruction in the above instruction set then result forwarding will not work.

12. What are the different pipeline hazards and what are the remedies?

[WBUT 2009, 2017]

OR,

What do you mean by pipeline hazards? What is a data hazard? What are several types of data hazards and the solutions of these?

[WBUT 2014, 2016]

Answer:

In computer architecture, a hazard is a potential problem that can happen in a pipelined processor. There are typically three types of hazards: data hazards, branching hazards, and structural hazards.

Instructions in a pipelined processor are performed in several stages, so that at any given time several instructions are being executed, and instructions may not be completed in the desired order. A hazard occurs when two or more of these simultaneous (possibly out of order) instructions conflict.

i. Data hazards

Data hazards occur when data is modified. Ignoring potential data hazards can result in race conditions. There are three situations a data hazard can occur in:

- **Read after Write (RAW):** An operand is modified and read soon after. Because the first instruction may not have finished writing to the operand, the second instruction may use incorrect data.
- **Write after Read (WAR):** Read an operand and write soon after to that same operand. Because the write may have finished before the read, the read instruction may incorrectly get the new written value.
- **Write after Write (WAW):** Two instructions that write to the same operand are performed. The first one issued may finish second, and therefore leave the operand with an incorrect data value.

The operands involved in data hazards can reside in memory or in a register.

ii. Structural hazards

A structural hazard occurs when a part of the processor's hardware is needed by two or more instructions at the same time. A structural hazard might occur, for instance, if a program were to execute a branch instruction followed by a computation instruction. Because they are executed in parallel, and because branching is typically slow (requiring a comparison, program counter-related computation, and writing to registers), it is quite possible (depending on architecture) that the computation instruction and the branch instruction will both require the ALU at the same time.

iii. Branch hazards

Branching hazards (also known as control hazards) occur when the processor is told to branch - i.e., if a certain condition is true, then jump from one part of the instruction stream to another - not necessarily to the next instruction sequentially. In such a case, the processor cannot tell in advance whether it should process the next instruction (when it may instead have to move to a distant instruction). This can result in the processor doing unwanted actions.

13. Use 8-bit 2's complement integer to perform $-43+(-13)$. **[WBUT 2009]**

Answer:

$-43 = 11010101$

$-13 = 11110011$

$-43 + (-13) = 111001000$

So, discard carry and the result is 11001000. i.e. the 2's complement of 56.

14. What do you mean by pipeline processing? **[WBUT 2009]**

Answer:

Pipelining refers to the technique in which a given task is divided into a number of subtasks that need to be performed in sequence. Each subtask is performed by a given functional unit. The units are connected in a serial fashion and all of them operate simultaneously. The use of pipelining improves the performance compared to the traditional sequential execution of tasks. The figure below shows an illustration of the basic difference between executing four subtasks of a given instruction (in this case fetching F, decoding D, execution E, and writing the results W) using pipelining and sequential processing.

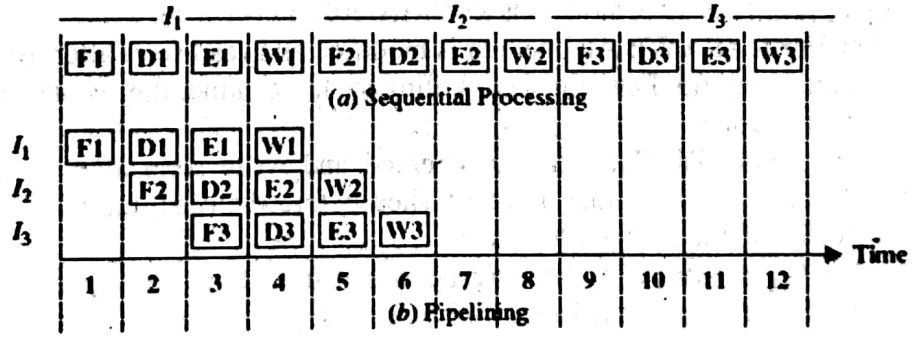


Fig: Pipeline vs sequential Processing

15. What are instruction pipeline and arithmetic pipeline? **[WBUT 2009]**

Answer:

An instruction pipeline is a technique used in the design of computers system to increase their instruction throughput. The fundamental idea is to split the processing of a computer instruction into a series of independent steps, with storage at the end of each step. The principles used in instruction pipelining can be used in order to improve the performance of computers in performing arithmetic operations such as add, subtract, and multiply.

In a program there is several numbers of instructions. There are five steps to execute an instruction and the steps are:

i. Fetch
ii. Decode
iii. Operand fetch
iv. Execute
v. Write back

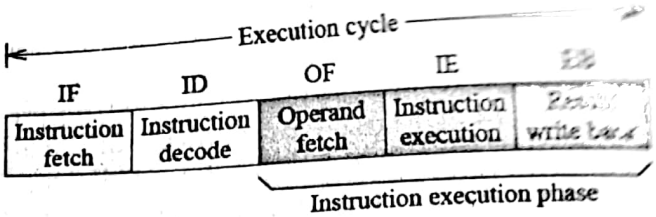
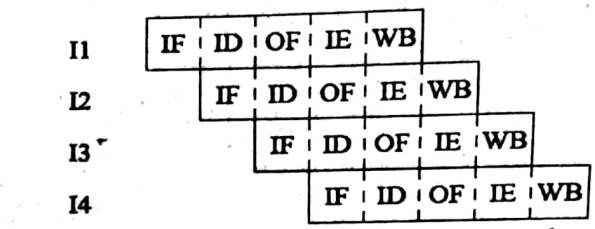


Figure: Instruction pipeline stages

The streams of instructions are executed in the pipeline in an overlapped manner.



Pipelining can be applied to arithmetic operations. As an example, we show a floating-point add pipeline in Figure below. The floating-point add unit has several stages:

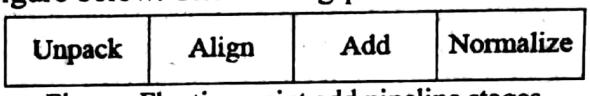


Figure: Floating point add pipeline stages

1. ***Unpack:*** The unpack stage partitions the floating-point numbers into the three field: the sign field, exponent field, and mantissa field. Any special cases such as not-a-number (NaN), zero, and infinities are detected during this stage.
2. ***Align:*** This stage aligns the binary points of the two mantissas by right-shifting the mantissa with the smaller exponent.
3. ***Add:*** This stage adds the two aligned mantissas.
4. ***Normalize:*** This stage packs the three fields of the result after normalization and rounding into the IEEE-754 floating-point format. Any output exceptions are detected during this stage.

16. Find 2's complement of $(1AB)_{16}$ represented in 16 bit format. [WBUT 2009]

Answer:

$(1AB)_{16} = (0000\ 0001\ 1010\ 1011)_2$

2's complement of $(0000\ 0001\ 1010\ 1011)_2$ is

$(1111\ 1110\ 0101\ 0101)_2$

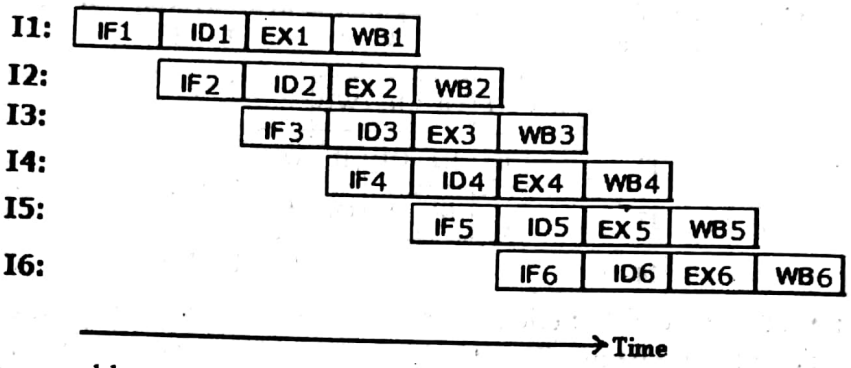
17. For the code segment given below, explain how delayed branching can help:

11. LOAD R1, A
12. Dec R3, 1
13. BrZero R3, 15
14. Add R2, R4
15. Sub R5, R6
16. Store R5, B

[WBUT 2015]

Answer:

Delayed branch is a conditional branch instruction includes in pipelining architecture. The effect is to execute one or more instructions following the conditional branch before the branch is taken. This avoids stalling the pipeline while the branch condition is evaluated, thus keeping the pipeline full and minimizing the effect of conditional branches on processor performance.



From the above problem, we get the following pipeline structure. The instruction I3 is the branch instruction and it creates the pipeline stall. To avoid this, we can apply delayed branch technique as follows:

11. LOAD R1, A
12. Dec R3, 1
13. Add R2, R4
14. Sub R5, R6
15. Store R5, B
16. BrZero R3, 15

The delayed branch can be applied before branch instruction from the target address. Here we use three stage delay i.e. the branch instruction will be executed at I6 not in I3.

18. What do you understand by instruction pipelining and arithmetic pipelining? Why pipeline scheduling is necessary and how it is done? [WBUT 2015]

Answer:

The pipeline structures used for instruction pipelining may be applied in some cases to other processing tasks. If pipelining is to be useful, however, we must be faced with the need to perform a long sequence of essentially similar tasks. Large numerical applications often make use of repeated arithmetic operations for processing the elements of vectors and arrays. Architectures specialized for applications if this type often provides pipelines to speed processing of floating-point arithmetic sequences. This type of pipelining is called arithmetic pipelining. Arithmetic pipelines differ from instruction pipelines in some important ways. They are generally synchronous. This means that each stage executes in a fixed number of clock cycles. In a synchronous pipeline, moreover, no buffering between stages is provided. Each stage must be ready to accept the data passed from a previous stage when that data is produced.

In order to speed up the operation of a computer system beyond what is possible with sequential execution, methods must be found to perform more than one task at a time. One method for gaining significant speedup with modest hardware cost is the technique of pipelining. In this technique, A task is broken down into multiple steps, and independent processing units are assigned to each step. Once a task has completed its initial step, another task may enter that step while the original task moves on to the following step. The first step in applying pipelining techniques to instruction processing is to divide the task into steps that may be performed with independent hardware. The most obvious division is between the FETCH cycle (fetch and interpret instructions) and the EXECUTE cycle (access operands and perform operation). Instruction scheduling is a compiler optimization used to improve instruction-level parallelism, which improves performance on machines with instruction pipelines.

Pipeline scheduling is necessary to avoid illegal or semantically ambiguous operations typically involving instruction pipeline timing issues or non-interlocked resources. The pipeline stalls can be caused by structural hazards (processor resource limit), data hazards (output of one instruction needed by another instruction) and control hazards (branching). To arrive at a good schedule, stalls should be prevented. This is determined by the choice of the next instruction to be scheduled. A number of heuristics are in common use:

- The processor resources used by the already scheduled instructions are recorded. If a candidate uses a resource that is occupied, its priority will drop.
- If a candidate is scheduled closer to its predecessors than the associated latency its priority will drop.
- If a candidate lies on the critical path of the graph, its priority will rise. This heuristic provides some form of look-ahead in an otherwise local decision process.
- If choosing a candidate will create many new sources, its priority will rise. This heuristic tends to generate more freedom for the scheduler.

19. Define pipelining technique. Assume a 4 stage pipeline:
Fetch: Read the instruction from the memory
Decode: Decode the instruction
Execute: Execute the instruction
Write: Store the result in destination location
Draw the space-time diagram for pipelining. **[WBUT 2015]**

Answer:

Pipelining was first applied to high performance supercomputers and is today the design paradigm for most micro architecture. The benefits of pipelining are: a faster clock and a theoretical execution rate of one instruction per clock, for degree one superscalar, while serial execution model processors have an execution rate of less than 0.1 instructions per clock. However, the high theoretical performance rate of a pipeline can be greatly diminished because of structural hazards, data hazards and branch stalls. Data hazards introduce stalls that can be mitigated by out of order instruction issue. Calling upon locality, designers implement reorder buffers to insure in order updating of the processor state. Branch stalls reduce pipeline performance leading to the need for branch prediction strategies and hardware.

Many prediction strategies depend on temporal locality employed with branch target buffers and branch target caches to benefit from the temporal locality of branch direction information.

Pipeline is an implementation technique whereby multiple instructions are overlapped in execution. Each step in the pipeline (called a *pipe stage*) completes a part of an instruction. Because all stages proceed at the same time, the length of a processor (clock) cycle is determined by the time required for the slowest pipe stage. Designer's goal: Balancing the length of each pipeline stage. If the stages are perfectly balanced, the time per instruction on the pipelined processor is,

$$\frac{\text{Time per instruction o un-pipelined machine}}{\text{Number of pipe stages}}$$

The Speedup from pipelining is equal to the number of pipe stages.

The implementation of an integer subset of RISC architecture takes at most 5 clock cycles.

- Instruction Fetch (IF)
- Instruction Decode/Register Fetch (ID)
- Execution/Effective Address Calculation (EX)
- Memory Access (MEM)
- Write-Back (WB)

Instruction Fetch Cycle (IF): In this cycle CPU sends the address which is held by program counter (PC) to the memory. Then fetch the current instruction from memory and update the content of PC for the next instruction.

Instruction Decode/Register Fetch Cycle (ID): In this cycle CPU decodes the instruction and reads the registers from the register file and performs the equality test on the registers for a possible branch. Then sign-extend the offset field of the instruction in case it is needed and computes the possible branch target address by adding the sign-extended offset to the incremented PC.

Execution/Effective Address Calculation (EX): In the execution cycle the ALU operates on the operands prepared in the prior cycle. If it is memory reference instructions, the ALU adds the base register and the offset to form the effective address or if it is Register-Register instruction then the ALU performs the operation specified by the ALU opcode on the values from the register file. If it is Register-Immediate instruction then the ALU performs the operation specified by the opcode on the first value from the register file and the sign-extended immediate.

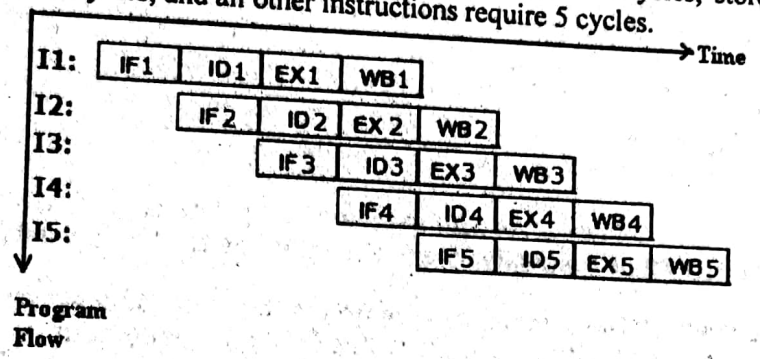
Memory Access (MEM)

- If the instruction is a load, memory does a read using the effective address computed in the previous cycle.
- If it is a store, then the memory writes the data from the second register read from the register file using the effective address.

Write-Back cycle (WB)

- Register-Register ALU instruction or Load instruction: Write the result into the register file.

In this implementation, branch instructions require 2 cycles, store instructions require 4 cycles, and all other instructions require 5 cycles.



20. What are in-order issue and out-order issue in superscalar pipeline? Describe with an example. **[WBUT 2016]**

Answer:

In-order instruction execution

- Instructions are fetched, executed & completed in compiler generated order
- One stalls, they all stall
- Instructions are statically scheduled

Out-of-order instruction execution

- Instructions are fetched in compiler-generated order
- Instruction completion may be in-order (today) or out-of-order (older computers)
- In between they may be executed in some other order
- Independent instructions behind a stalled instruction can pass it
- Instructions are dynamically scheduled

Long Answer Type Questions

1. a) Distinguish a static pipeline from a dynamic pipeline.

b) Show that average latency of any greedy cycle ≤ number of 1's in the initial collision vector. **[WBUT 2006]**

Answer:

a) In general, a static pipeline is one, which follows a fixed series of steps. Traditional linear pipeline are static pipelines because they are used to perform fixed functions. The given function is partitioned into a sequence of linearly ordered static sub functions. In CPU design, static pipelines are easier to design and less expensive to manufacture. The alternative is a dynamic pipeline, where the steps are determined by a set of rules. A dynamic pipeline can be reconfigured to perform variable functions at different times. It allows feed forward and feed back connections in addition to streamline connections. In CPU design, dynamic pipelines can significantly boost performance with out of order execution.

b) The average latency of a latency cycle is obtained by dividing the sum of all latencies by the number of latencies along the cycle. A greedy cycle is one whose edges are all

made with minimum latencies from their respective starting states. The greedy cycles are simple and their average latencies must be lower than those of other simple cycle.
Now,
The maximum number of marks in any row of the reservation table is ≤ MAL and Minimal Average Latency (MAL) is ≤ average latency of any greedy cycle.
So, the average latency of any greedy cycle ≤ number of 1's in the initial collision vector.

2. Suppose the time delays of the four stages of a pipeline are τ_1 = 60 ns, τ_2 = 50 ns, τ_3 = 90 ns and τ_4 = 80 ns respectively and the interface latch has a delay τ_1 = 10 ns, then

i) **what would be the maximum clock frequency of the above pipeline?**
ii) **what is the maximum speed-up of this pipeline over that of its non-pipeline counterpart?** **[WBUT 2006]**

Answer:

i) The cycle time of this pipeline can be chosen at least $\tau = (90+10)\,ns = 100\,ns$.

∴ The clock frequency of the pipeline can be set to

$$f = 1/\tau = 1/100 = 10 \text{ MHz.}$$

ii) If non-pipelined floating-point adder is used, then total time delay will be $\tau_1 + \tau_2 + \tau_3 + \tau_4 = 300\,ns$

The pipeline adder has a speedup of $300/100 = 3$ over non-pipeline counterpart.

3. Describe the time stationary and data stationary control schemes. [WBUT 2006]

Answer:

A control scheme determines how a segment decides what operation to perform on each transaction and where to send the transaction when it is done in the segment. The specification for the control scheme says that if every transaction goes to the correct segments and each segment performs the correct operation, then every transaction will eventually request to exit the pipeline. A segment can decide what operation it is to perform on a transaction based on ***data-stationary*** and/or ***time-stationary*** control information. Data-stationary control information is carried along with the transaction as it proceeds between segments. Time-stationary information is provided to the segment at each time cycle, regardless of what transaction is in the segment. A segment can determine where to send a transaction when it is done based on the data-stationary information with which the transaction entered the segment, data-dependent information generated within the segment (such as the result of a comparison or an overflow condition), and/or the current value of time-stationary control signals.
Each ordering scheme defines a function that takes two times (t_1 and t_2) and returns true if and only if the transaction exiting the pipeline at t_2 resulted from the transaction entering the pipeline at t_1. The ordering scheme is only used for verification; it is not implemented in hardware. Ordering is important for verification, because we want to ensure that all transactions produced by the pipeline have the correct data.
For pipelines with in-order execution, examining the request and accept signals between the pipeline and the environment is sufficient to determine the correspondence between

input and output transactions. If transactions can bypass segments, then they might exit the pipeline in a different order than they were issued. In these situations the destination of the transaction (e.g. the register in which the result will be stored) can many times act as a tag with which transactions are ordered. That is, transactions with the same tag exit the pipeline in the order which they were issued.

4. What is a pipeline?

Consider the following reservation table: **[WBUT 2007, 2011, 2012]**

	1	2	3	4
S1	X			
S2		X		X
S3			X	

Write down the forbidden latencies and initial collision vector. Draw the state diagram for scheduling the pipeline. Find out the sample and greedy cycle and MAL. If the pipeline clock rate is 25 MHz, then what is the throughput of the pipeline? What are the bounds on MAL?

Answer:

1st Part:

Pipelining is a technique of splitting a sequential process into sub operations being execute of different segment that operates concurrently with all other segments. An instruction pipeline is a technique used in the design of computers to increase their instruction throughput (the number of instructions that can be executed in a unit of time). Pipelining assumes that with a single instruction (SIMD) concept successive instructions in a program sequence will overlap in execution.

A non-pipeline architecture is inefficient because some CPU components are idle while another module is active during the instruction cycle. Pipelining does not completely cancel out idle time in a CPU but making those modules work in parallel improves program execution significantly.

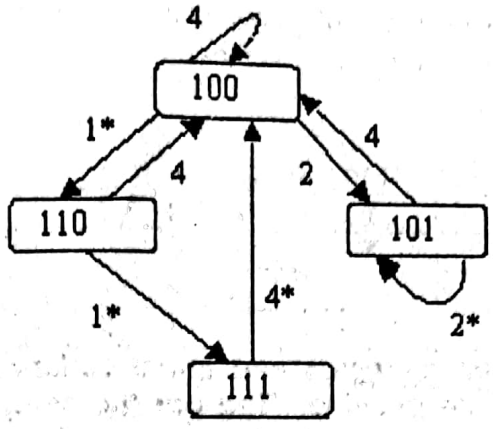
Processors with pipelining are organized inside into stages which can semi-independently work on separate jobs. Each stage is organized and linked into a 'chain' so each stage's output is inputted to another stage until the job is done. This organization of the processor allows overall processing time to be significantly reduced.

Unfortunately, not all instructions are independent. In a simple pipeline, completing an instruction may require 5 stages. To operate at full performance, this pipeline will need to run 4 subsequent independent instructions while the first is completing. If 4 instructions that do not depend on the output of the first instruction are not available, the pipeline control logic must insert a stall or wasted clock cycle into the pipeline until the dependency is resolved. Fortunately, techniques such as forwarding can significantly reduce the cases where stalling is required. While pipelining can in theory increase performance over an unpipelined core by a factor of the number of stages (assuming the clock frequency also scales with the number of stages), in reality, most code does not allow for ideal execution.

2nd Part:

Forbidden latencies means the latencies that cause collision. Here the forbidden latency

is 3.
The initial collision vector is 100.
The state diagram is given below



Simple cycle: (2), (4), (1,4), (1,1,4) and (2,4)
Greedy cycle: (2)
MAL = 2
Throughput = 2 MIPS
Upperbound = 2
Lowerbound = 2

5. a) What do you mean by "Data flow Computer"? **[WBUT 2008]**
b) With simple diagram, explain Data flow architecture and compare it with control flow architecture. **[WBUT 2008]**

OR,

Explain data flow architectures. **[WBUT 2008, 2012]**
c) Draw data flow graphs to represent the following computations:
i) X=A+B
ii) Y=X/B
iii) Z=A*X
iv) M=Z–Y
v) N=Z*X
vi) P=M/N. **[WBUT 2008]**

Answer:

a) Data flow computer is a large, very powerful computer that has a number of processors all physically wired together with a large amount of memory and backing storage. Such computers are highly parallel in that they can carry out a large number of tasks at the same time. Data flow computers are used to execute processor intensive applications such as those associated with areas like molecular biology and simulation. Numerical calculations for the simulation of natural phenomena were conducted using a data-flow-type parallel processing computer. The computing time in the data-flow computer was approximately three-to-five times shorter than that of the usual medium-size computer of computing speed 3 MIPS (million instructions per second). Dynamic visualization of the computing process was realized using an image display directly connected to the memory of the data-flow computer.

b) The Control-Flow Architecture is a Von Neumann or control flow computing model. Here a program is a series of addressable instructions, each of which either specifies an operation along with memory locations of the operands or it specifies conditional transfer of control to some other instruction. The next instruction to be executed depends on what happened during the execution of the current instruction. The next instruction to be executed is pointed to and triggered by the PC. The instruction is executed even if some of its operands are not available yet.

But in Dataflow model, the execution is driven only by the availability of operand. There is no Program Counter and global updateable store. The two features of von Neumann model that become bottlenecks in exploiting parallelism are missing in Data flow Architecture.

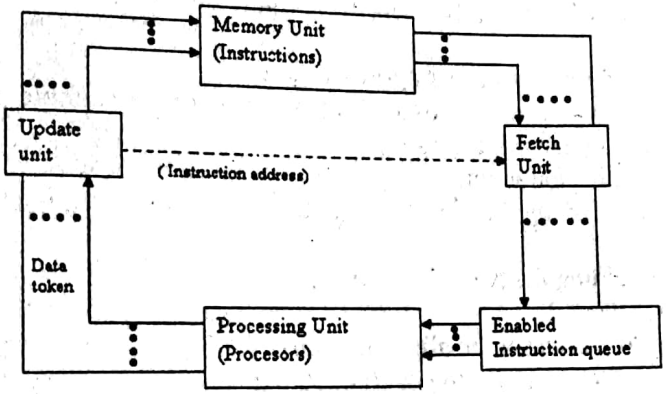


Fig: A static Data flow Computer

c)

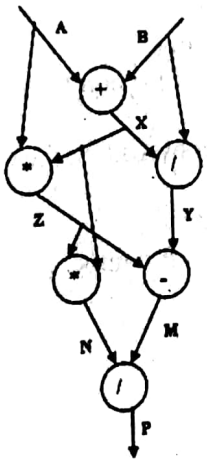


Fig: Data Flow graph for the above computation

6. What is floating point arithmetic operation? Explain all (addition, difference, multiplication, division) operations with example. **[WBUT 2009]**

Answer:

A floating-point (FP) number can be represented in the following form: $\pm m * b^e$

Where m is the mantissa and it represents the fraction part of the number and is normally represented as a signed binary fraction, e represents the exponent, and b represents the base (radix) of the exponent.

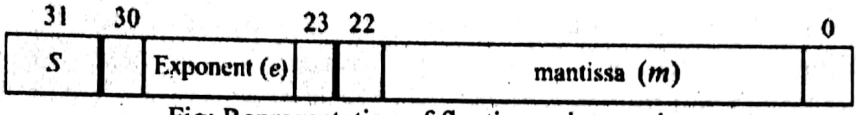


Fig: Representation of floating point number

Floating-Point Arithmetic Addition/Subtraction:

The difficulty in adding two FP numbers stems from the fact that they may have different exponents. Therefore, before adding two FP numbers, their exponents must be equalized, that is, the mantissa of the number that has smaller magnitude of exponent must be aligned.

Steps required add/subtract two Floating-Point numbers:

1. Compare the magnitude of the two exponents and make suitable alignment to the number with the smaller magnitude of exponent.
2. Perform the addition/subtraction.
3. Perform normalization by shifting the resulting mantissa and adjusting the resulting exponent.

Example: Consider adding the two FP numbers:

1. $1100 * 2^4$ and $1.1000 * 2^2$.
1. Alignment: $1.1000 * 2^2$ has to be aligned to $0.0110 * 2^4$
2. Addition: Add the two numbers to get $10.0010 * 2^4$.
3. Normalization: The final normalized result is $1.0001 * 2^5$ (assuming 4 bits are allowed after the radix point).

Floating-Point Arithmetic Multiplication

A general algorithm for multiplication of FP numbers consists of three basic steps. These are:

1. Compute the exponent of the product by adding the exponents together.
2. Multiply the two mantissas.
3. Normalize and round the final product.

Example Consider multiplying the two FP numbers

$X = 1.000 * 2^{-2}$ and $Y = -1.010 * 2^{-1}$

1. Add exponents: $-2 + (-1) = -3$.
2. Multiply mantissas: $1.000 * - 1.010 = -1.010000$.

The product is $-1.0100 * 2^{-3}$.

Floating-Point Arithmetic Division

A general algorithm for division of FP numbers consists of three basic steps:

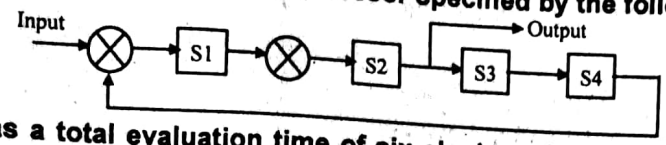
1. Compute the exponent of the result by subtracting the exponents.
2. Divide the mantissa and determine the sign of the result.
3. Normalize and round the resulting value, if necessary.

Consider the division of the two FP numbers

$X = 1.0000 * 2^{-2}$ and $Y = -1.0100 * 2^{-1}$.

1. Subtract exponents: -2 - (-1) = -1.
2. Divide the mantissas: (1.0000) / (-1.0100) = -0.1101.
3. The result is $-0.1101 * 2^{-1}$.

7. Consider the four stage pipelined processor specified by the following diagram:



The pipeline has a total evaluation time of six clock cycles. All successor stages must be used after each clock cycle.

i) Specify the reservation table for above pipelined processor with six columns and four rows.

ii) What are the forbidden latencies and the initial collision vector? Draw the state transition diagram.

iii) Determine all simple cycles, greedy cycle and MAL.

iv) Determine the throughput of this pipelined processor. Given clock period as 20 ns.

[WBUT 2010]

Answer:

i)

	1	2	3	4	5	6
S1	X				X	
S2		X				X
S3			X			
S4				X		

ii) The forbidden Latency is (0,4)
The pipeline collision vector (100010)

State 1 : 100010
Reaches State 2 (100110) after 1 cycle.
Reaches State 3 (101010) after 2 cycles.
Reaches State 4 (110010) after 3 cycles.
Reaches State 1 (100010) after 5 cycles.
Reaches State 1 (100010) after 6 or more cycles.

State 2 : 100110
Reaches State 5 (101110) after 1 cycle.
Reaches State 6 (111010) after 2 cycles.
Reaches State 1 (100010) after 5 cycles.
Reaches State 1 (100010) after 6 or more cycles.

State 3 : 101010
Reaches State 7 (110110) after 1 cycle.
Reaches State 4 (110010) after 3 cycles.
Reaches State 1 (100010) after 5 cycles.
Reaches State 1 (100010) after 6 or more cycles.

State 4 : 110010
Reaches State 3 (101010) after 2 cycles.
Reaches State 4 (110010) after 3 cycles.
Reaches State 1 (100010) after 5 cycles.
Reaches State 1 (100010) after 6 or more cycles.

State 5 : 101110
Reaches State 8 (111110) after 1 cycle.
Reaches State 1 (100010) after 5 cycles.
Reaches State 1 (100010) after 6 or more cycles.

State 6 : 111010
Reaches State 4 (110010) after 3 cycles.
Reaches State 1 (100010) after 5 cycles.
Reaches State 1 (100010) after 6 or more cycles.

State 7 : 110110
Reaches State 6 (111010) after 2 cycles.
Reaches State 1 (100010) after 5 cycles.
Reaches State 1 (100010) after 6 or more cycles.

State 8 : 111110
Reaches State 1 (100010) after 5 cycles.
Reaches State 1 (100010) after 6 or more cycles.

There are 8 states in the state diagram.

iii) Greedy cycle: (1↓15)

MAL = 2

iv) The throughput of this pipelined processor = (1/20)x (1/2) = 0.025 instructions per cycle.

8. a) What do you mean by MMX? Differentiate a data flow computer from a control flow computer.
b) List some potential problems with data flow computer implementation.
c) With simple diagram explain data flow architecture.
d) Draw data flow graphs to represent the following computations:

i) $P = X + Y$ **ii)** $Q = P \div Y$
iii) $R = X * P$ **iv)** $S = R - Q$
v) $T = R * P$ **vi)** $U = S \div T$ **[WBUT 2011]**

OR,

Draw data flow graph to represent the following computations:

1. $A = P + Q$ **2.** $B = A / Q$
3. $C = P * A$ **4.** $D = C - B$
5. $E = C * A$ **6.** $F = D / E$ **[WBUT 2015]**

Answer:

a) MMX technology is an extension to the Intel Architecture (IA) designed to improve performance of multimedia and communication algorithms. The Pentium processor with MMX Technology is the first microprocessor to implement the new instruction set. All MMX chips have a larger internal L1 cache than their non-MMX counterparts. The Pentium processor with MMX implementation was the design of a new, dedicated, high-performance MMX pipeline, which was able to execute two MMX instructions with minimal logic changes in the existing units. Although adding a pipeline stage improves frequency, it decreases CPI performance, i.e., the longer the pipeline, the more work done speculatively by the machine and therefore more work is being thrown away in the case of branch miss-prediction.

The control flow computers are either uniprocessor or parallel processors architecture. In uniprocessor system the instructions are executed sequentially and this is called control-driven mechanism. In parallel processors system control flow computers use shared memory. So, parallel executed instructions may cause side effects on other instructions and data due to shared memory. In control flow computer the sequence of execution of instructions is controlled by program counter register.

The data flow computers are based on a data driven mechanism. The fundamental difference is that instruction execution in a conventional computer is under program-flow control, whereas that in a data flow computer is driven by the data (operand) availability.

b) The data driven mechanism not requires any shared memory, program counter and control sequencer. It just checks data availability of needy instructions and to enable the chain reaction of asynchronous instruction execution.

c) Suppose, there are some instructions given below. Now we execute these instructions by a data flow machine.

Input: a, b, c

$k_0 = 0$

for i = 1 to 8 do

begin

$d_i = a_i + b_i$

$e_i = d_i * c_i$

$k_i = e_i + k_{i-1}$

end

Output d, e, k.

In the above example, there are three instructions in the "for loop" and the "for loop" is executed 8 times. So, total 24 instructions will be executed. Suppose, each add, multiply and divide operation requires 1, 2 and 3 clock cycles to complete respectively. The data flow graph is shown in the figure below, for above instructions.

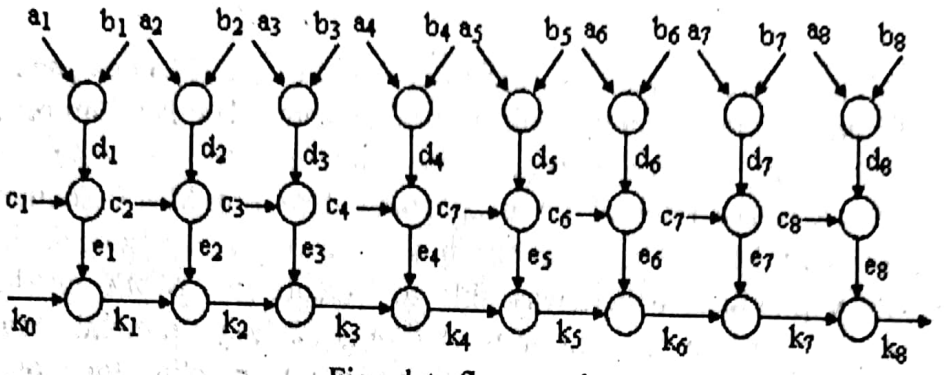


Fig: data flow graph

The above instructions can be executed within 14 clock cycles as shown in the figure below.

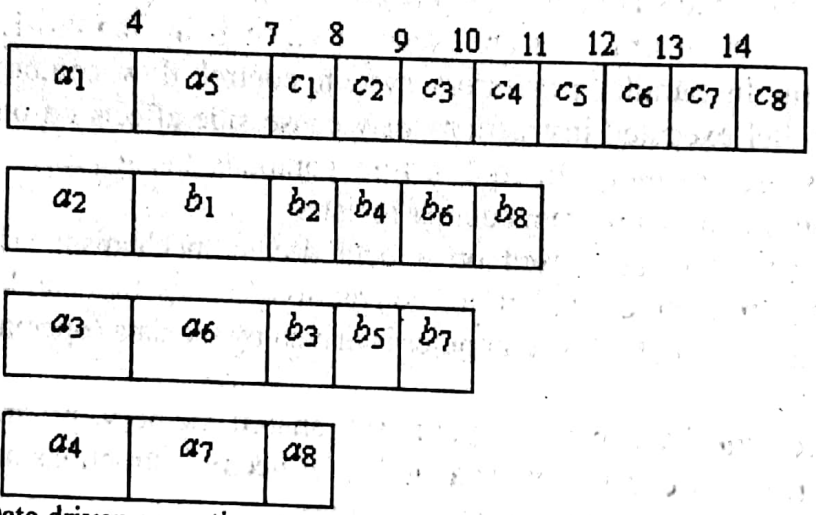
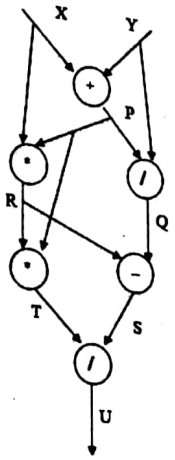


Fig: Data driven execution on a 4 processor dataflow computer in 14 cycles

The dataflow multiprocessor completes the execution in 14 cycles. If all the external inputs are available, instructions a_1, a_2, a_3 and a_4 are all ready for execution in the first three cycles produced then trigger the execution of a_5, b_1, a_6, and a_7 starting from cycle 4. The data-driven chain reactions are shown in figure above. The output c8 is the last one to produce, due to its dependence on all the previous c_i's.
The theoretical minimum time is 13 cycles along the critical path $a_1b_1c_1c_2...c_8$. The chain reaction control in dataflow is more difficult to implement and may result in longer overhead, as compared with the uniform operations performed by all - processors.

d)



9. a) **What is reservation table?**
b) **Consider a five-stage pipelined processor specified by following reservation table:**

	1	2	3	4	5	6
S_1	×					×
S_2		×			×	
S_3			×			
S_4				×		
S_5		×				×

i) List the set of forbidden latencies and the collision vector.
ii) Draw the state transition diagram.
iii) List all simple cycles from state diagram.
iv) Identify the greedy cycles among the simple cycles
v) Find out minimum Average latency.
vi) find out maximum throughput of these pipeline.
c) Show that the maximum speed-up of a pipeline is equal to its number of stages. **[WBUT 2013]**

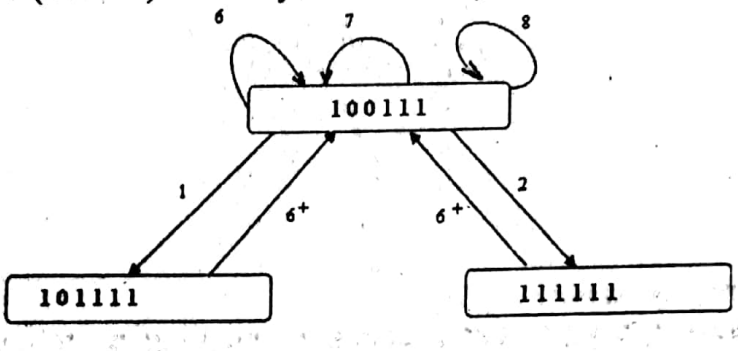
Answer:
a) A reservation table is a two-dimensional table in which each row represents a pipeline stage (processing element) and each column represents a cycle in the execution of a task. Within each column, an X is placed in the entries corresponding to the stages required by that task during that cycle.

b) i) The forbidden latencies, F= 0,3, 4, 5
The initial collision vector, V= [1,0,0,1,1,1]
ii) State 1: 100111
Reaches state 2 (101111) after 1 cycle
Reaches state 3 (111111) after 2 cycle

Reaches state 1 (100111) after 6 cycle or more cycles

State 2: 101111
Reaches state 3 (111111) after 1 cycle
Reaches state 1 (100111) after 6 cycle or more cycles

State 3: 111111
Reaches state 1 (100111) after 6 cycle or more cycles



iii) The simple cycles are : (1,6) , (2,6), (6), (7), (8), (1,6,6), (1,6,7), (1,6,8), (2,6,6), (2,6,7), (2,6,8)
iv) The Greedy cycle is: (1,1,6)
v) MAL= 2.66
vi) The maximum throughput = 0.375 instruction per cycle

c) A linear pipeline of k stages can process n tasks in k + (n–1) clock cycles, where k cycles are needed to complete the execution of the very first task and the remaining n–1 tasks require n–1 cycles. Thus the total time required is

$$T_k = [k + (n-1)]\tau$$

where τ is the clock period. Consider an equivalent-function non pipelined processor which has a flow-through delay of $k\tau$. The amount of time it takes to execute n tasks on this non pipelined processor is $T_1 = nk\tau$.

Speedup Factor The speedup factor of a k-stage pipeline over an equivalent non pipelined processor is defined as

$S_k = T_1 / T_k = nk\tau / [k + (n-1)]\tau = nk / (k + n-1)$.

and in the limit, as the number of data items tends to infinity

$$\lim_{n \to \infty} \frac{T_1}{T_k} = k$$

This defines the asymptotic maximum speed-up, in terms of throughput, and this is equal to the number of pipeline stages. In general, of course, the flow of data items in any real pipeline is subject to disruptions of one kind or another and maximum throughput cannot be maintained continuously.

10. a) What is the difference between computer organization and computer architecture?

b) What is meant by pipeline architecture? How does it improve the speed of execution of a processor?

c) Consider the following reservation table for a 4-stage pipeline with a clock cycle $t = 20$ ns.

	1	2	3	4	5	6	
	×						
		×				×	S1
			×				S2
				×			S3
					×		S4

i) What are the forbidden latencies and the initial collision vector?

ii) Draw the state transition diagram for scheduling the pipeline.

iii) Determine the MAL associated with the shortest greedy cycle.

iv) Determine the pipeline throughput corresponding to the MAL and given t.

v) Determine the lower bound on the MAL for this pipeline.

Answer: **[WBUT 2014, 2016]**

a) Computer Organization:

Computer Organization encompasses all the physical aspects of computer system. It will answer you how does a computer work. Example: circuit design, control signals, memory types this all are under computer organization.

Computer architecture: The visual interface of computer. Logical terms as seen by the programmer. Computer architecture will answer you how do I design a computer. Example: instruction sets, instruction formats, data types, addressing modes.

b) ***Pipeline throughput***

The throughput of a pipeline is defined as the number of instructions that can be executed per time unit. All stages precede in synchronized fashion they all start at the same times (simplifies hardware design). The time required for moving one instruction down the pipeline is called a processor cycle (not to be confused with clock cycle). Because all pipe stages must be ready to proceed synchronously, the processor cycle is determined by the slowest stage.

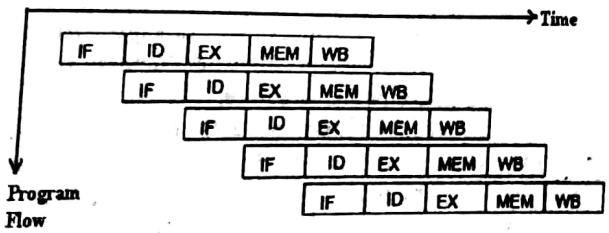


Fig: 1 Pipeline technique

Performance of the pipeline

Pipelining achieves a reduction of the average execution time per instruction. This is in the sense that one can perform more instructions per clock cycle. This can be viewed in two ways:

- Decreasing the CPI. Typical way in which people view the performance increase
- Decreasing the cycle time (i.e., increasing the clock rate). The good news is that pipelining is typically invisible to the programmer.

Pipelining increases the CPU instruction throughput. Pipelining does not decrease the execution time of an individual instruction. It increases the execution time due to overhead (clock skew and pipeline register delay) in the control of the pipeline.

Example:

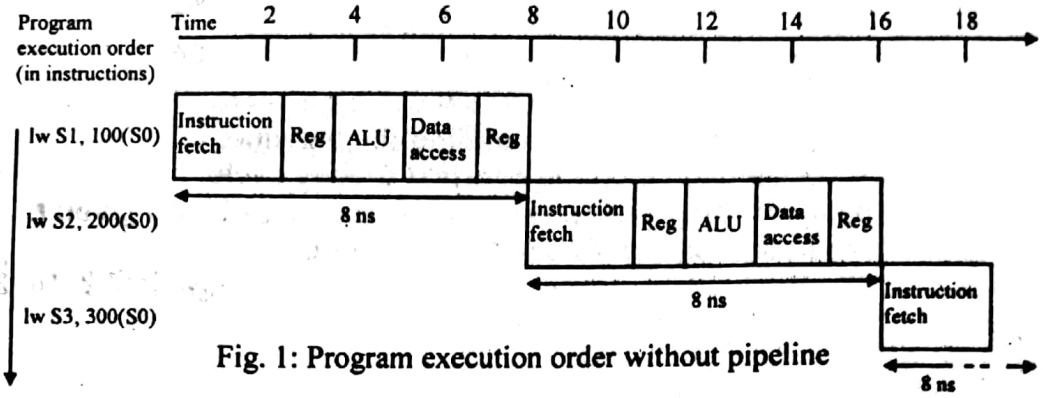


Fig. 1: Program execution order without pipeline

In the figure 1, we have shown non pipeline architecture. In this situation each instruction require 8ns and total time require to execute these three instructions is $8 \times 3 = 24$ns.

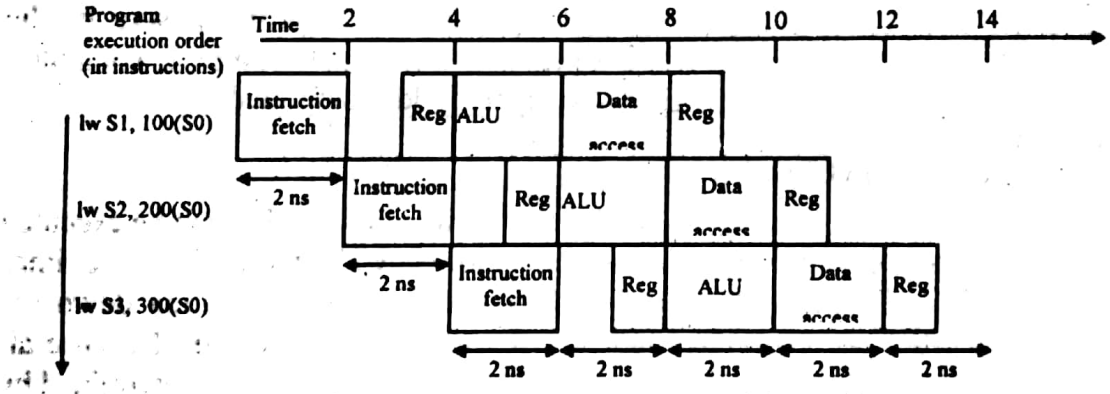


Fig. 2: Program execution order with pipeline

Now, we can execute these above three instructions in pipeline manner as describe in figure 2. In this case time requires to execute three instructions is $3 \times 2 = 6$ns. So, pipeline structure is faster than a non-pipeline structure.

A linear pipeline of k stages can process n tasks in $k + (n-1)$ clock cycles, where k cycles are needed to complete the execution of the very first task and the remaining n–1 tasks require n–1 cycles. Thus the total time required is $T_k = [k + (n-1)]\tau$

where τ is the clock period. Consider an equivalent-function non pipelined processor which has a flow-through delay of $k\tau$. The amount of time it takes to execute n tasks on this non pipelined processor is $T_1 = nk\tau$.

Speedup Factor The speedup factor of a k-stage pipeline over an equivalent non pipelined processor is defined as

$S_k = T_1 / T_k = nk\tau / [k + (n-1)]\tau = nk / (k + n-1)$.

c) (i) The Forbidden Latencies = 0, 1, 2, 5
The Pipeline Collision Vector = (1 1 1 0 0 1)

(ii) State transition diagram for scheduling the pipeline
State 1 : 111001
Reaches State 1 (111001) after 3 cycles.
Reaches State 1 (111001) after 4 cycles.
Reaches State 1 (111001) after 6 or more cycles.
There are 1 state in the state diagram:
State 1 represents 111001

(iii) The Minimum Average Latency=3 and the Greedy Cycle is (3)*

(iv) Throughput
$W = (\eta / T) * (1/ MAL)$
$= (1/ 20 \times 10^{-9}) * (1/ 3) = 16.67$ MIPS

(v) The MAL is lower-bounded by the maximum number of checkmarks in any row of the reservation table. So, the lower bound on the MAL for this pipeline = 2.

11. a) What is the arithmetic pipeline? Describe with an example.
b) What is register tagging or data forwarding? Describe several methods of it.
[WBUT 2014, 2016]

Answer:

a) The pipeline structures used for instruction pipelining may be applied in some cases to other processing tasks. If pipelining is to be useful, however, we must be faced with the need to perform a long sequence of essentially similar tasks. Large numerical applications often make use of repeated arithmetic operations for processing the elements of vectors and arrays. Architectures specialized for applications if this type often provides pipelines to speed processing of floating-point arithmetic sequences. This type of pipelining is called arithmetic pipelining.

Arithmetic pipelines differ from instruction pipelines in some important ways. They are generally synchronous. This means that each stage executes in a fixed number of clock cycles. In a synchronous pipeline, moreover, no buffering between stages is provided. Each stage must be ready to accept the data passed from a previous stage when that data is produced.

As an example of a pipelined arithmetic unit we consider a floating point adder. This pipeline accepts as input two normalized floating point numbers of the form:

$A = a \times 2^p$
$B = b \times 2^q$

Here a and b are 2's complement fractions in the range $0.5<f<1.0$. p and q are corresponding base 2 exponents. The normalized sum is to be computed. Four stages can be identified for this pipeline:

1. Input the original fractions and exponents. Compute the larger exponent and the exponent difference. Shift the fraction corresponding to the smaller exponent right for a number of places equal to the difference. Both fractions are now adjusted to match the same (larger) exponent. Output the exponent and the two fractions.

2. Add the two fractions, producing a sum. Pass through the exponent unchanged. Output the exponent and fractions.
3. Count leading zeros in the result fraction. Shift the fraction to normalize. Output the original exponent, the fraction, and the count.
4. Add the exponent and count. Output the adjusted exponent and the normalized fraction.

b) Internal Forwarding: It is replacing unnecessary memory accesses by register-to-register transfers. Memory access is slower than register-to-register operations and the performance can be enhanced by eliminating unnecessary memory accesses. This concept can be explored in 3 directions:

- Store – Load Forwarding
- Load – Load Forwarding
- Store – Store Forwarding

Register Tagging: It is the use of tagged registers for exploiting concurrent activities among multiple ALUs. Example: IBM Model 91 (Floating Point Execution Unit). In that system, the two pairs at the input end of the multiply / divide unit made it behave like two virtual units. Internal data forwarding in Model 91 was accomplished using source tags on all registers and reservation stations. Divide was implemented in Model 91 based on the convergence method. Every source of an input operand was uniquely identified with a 4-bit tag. Every destination of an input operand had an associated tag register that held the tag naming the source of data if the destination was busy. Through this register tagging technique, operand / results could be directly passed among the virtual functional units. This forwarding significantly cut down the data flow time between them.

12.a) Differentiate between instruction level parallelisms and thread level parallelism.
b) What are data hazards that are noticed in instructing level parallelism?
[WBUT 2017]

Answer:

a) Instruction level parallelism has the ability of CPU to execute more than one instruction simultaneously. This means that at each pipeline stage there are 2 or more instructions executing in parallel. This is implemented on a hardware level by having 2 or more instruction decoders and 2 or more ALU's depending on level of parallelism required. Although the processor may support instruction level parallelism it doesn't mean that for a given program under execution all the instructions will execute in parallel. Because the instructions may share data, it might not be possible to execute instructions which try to operate on and change same data at same time. Machine parallelism is a measure of ability of processor to take advantage of instruction level parallelism.

b) In different pipeline techniques, instruction-level parallelism (ILP) is based on the idea of multiple issue processors (MIP). An MIP has multiple pipelined data-paths for instruction execution. Instructions in a pipelined processor are performed in several

stages, so that at any given time several instructions are being executed, and instructions may not be completed in the desired order. A hazard occurs when two or more of these simultaneous (possibly out of order) instructions conflict. Data hazards occur when data is modified. Ignoring potential data hazards can result in race conditions. There are three situations a data hazard can occur in:

•Read after Write (RAW): An operand is modified and read soon after. Because the first instruction may not have finished writing to the operand, the second instruction may use incorrect data.

•Write after Read (WAR): Read an operand and write soon after to that same operand. Because the write may have finished before the read, the read instruction may incorrectly get the new written value.

•Write after Write (WAW): Two instructions that write to the same operand are performed. The first one issued may finish second, and therefore leave the operand with an incorrect data value.

The operands involved in data hazards can reside in memory or in a register.

13. a) What is ILP?
b) What are the two major approaches for ILP?
c) Differentiate between static and dynamic branch prediction approaches?
d) What is VLIW? List out the advantage of VLIW?
e) What is the major difference between superscalar and VLIW processors?
f) What is the limitation of VLIW processors?

[WBUT 2017]

Answer:

a) All processors use pipelining to overlap the execution of instructions and improve performance. This potential overlap among instructions is called *instruction-level parallelism* (ILP) since the instructions can be evaluated in parallel. The amount of parallelism available within a basic block–a straight line code sequence with no branches in except to the entry and no branches out except at the exit–is quite small. For typical MIPS programs the average dynamic branch frequency often between 15% and 25%, meaning that between four and seven instructions execute between a pair of branches. Since these instructions are likely to depend upon one another, the amount of overlap we can exploit within a basic block is likely to be much less than the average basic blocks size. To obtain substantial performance enhancements, we must exploit ILP across multiple basic blocks.

b) There are two largely separable approaches to exploiting ILP. 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. Here is a simple example of a loop, which adds two 1000-element arrays, that is completely parallel:

```
for (i=1; i<=1000; i=i+1)
x[i] = x[i] + y[i];
```

Iterations of the loop can overlap with any other iteration, although within the loop there is little or no opportunity for overlap.

An important alternative method for exploiting loop-level parallelism is the use of vector instructions. Essentially, a vector instruction operates on a sequence of data items. For example, the above code sequence could execute in four instructions on some vector processors: two instructions to load the vectors x and y from memory, one instruction to add the two vectors, and an instruction to store back the result vector. Of course, these instructions would be pipelined and have relatively long latencies, but these latencies may be overlapped.

c) Static Branch Prediction

- Previously scheduled code around delayed branch
- To reorder code around branches
- Need to predict branch statically during compile
- Simplest scheme is to predict a branch as taken

Dynamic Branch Prediction

–algorithm has monotonies

– Data that is being operated on has monotonies

– Instruction sequence has redundancies that are artifacts of way that humans/compilers think about problems

-There are a small number of important branches in programs that have dynamic behavior.

d), e) & f) ***Refer to Question No. 9 of Short Answer Type Questions.***

14. Write short notes on the following:

a) Reservation Table **[WBUT 2008]**

b) Pipeline hazards **[WBUT 2011, 2013]**

c) Branch handling in instruction pipeline **[WBUT 2011]**

d) Data flow computer **[WBUT 2014, 2016]**

Answer:

a) Scheduling and control are important factors in the design of nonlinear and dynamic pipelines. Any time the hardware is reconfigured, or any time the same stage is used more than once in a computation, a structural hazard may exist, meaning there is the possibility of a *collision* in the pipeline. A collision is an attempt to use the same stage for two or more operations at the same time. If two or more sets of inputs arrive at the same stage simultaneously, at the very least the pipeline will compute erroneous results for at least one set of inputs. Depending on the details of physical construction, the outputs of different stages could even be short-circuited to a common input, possibly causing damage to the circuitry. Thus collisions are to be avoided at all costs when controlling a pipeline.

How can we determine when collisions might occur in a pipeline? One graphical tool we can use to analyze pipeline operation is called a *reservation table.* A reservation table is just a chart with rows representing pipeline stages and columns representing time steps

(clock cycles). Marks are placed in cells of the table to indicate whi
pipeline are in use at which time steps while a given computation is bein
simplest reservation table is one for a static, linear pipeline. Table 1 is an example
reservation table for a five-stage static linear pipeline.

Notice that all the marks in this simple reservation table lie in a diagonal line. This is because each stage is used once and only once, in numerical order, in performing each computation. Even if we permuted the order of the stages, there would be still be only one mark in each row because no stage would be used more than once per operation. As long as this is the case, we will be able to initiate a new operation in the pipeline on each clock cycle. Suppose instead that we had a nonlinear pipeline. Some of the stages are used more than once per computation. This pipeline would have a reservation table as depicted in Table 2. Notice that in this case rows 2 and 3 of the table contain more than one mark, depicting the repeated use of the corresponding stages.

	t_0	t_1	t_2	t_3	t_4
Stage 1	X				
Stage 2		X			
Stage 3			X		
Stage 4				X	
Stage 5					X

Table 1: Reservation table for static linear pipeline

	t_0	t_1	t_2	t_3	t_4
Stage 1	X				
Stage 2		X		X	
Stage 3			X		X

Table 2: Reservation table for non linear pipeline

Latches are used between pipeline stages to get Minimum Average Latency (MAL). An optimization technique based on introducing a method to search the most proper location of no compute delay latches between nonlinear pipeline stages is given. The idea is to find a new collision vector which is adaptable with pipeline topology and modifies reservation table, yielding MAL at minimum execution time. This approach not only reduces execution time of hardware, but also minimizes favorite collision vector search time.

b) ***Refer to Question No. 12 of Short Answer Type Questions.***

c) In pipeline process the branch instructions are those that tell the processor to make a decision about what the next instruction to be executed should be based on the results of another instruction. Branch instructions can be troublesome in a pipeline if a branch is conditional on the results of an instruction which has not yet finished its path through the pipeline.

For example:

```
Loop : add   $r3,   $r2,   $r1
       sub   $r6,   $r5,   $r4
       equal $r3, $r6,  Loop
```

The example above instructs the processor to add r1 and r2 and put the result in r3, then subtract r4 from r5, storing the difference in r6. In the third instruction, beq stands for branch if equal. If the contents of r3 and r6 are equal, the processor should execute the instruction labeled "Loop." Otherwise, it should continue to the next instruction. In this example, the processor cannot make a decision about which branch to take because neither the value of r3 or r6 have been written into the registers yet.

The processor could stall, but a more sophisticated method of dealing with branch instructions is branch prediction. The processor makes a guess about which path to take - if the guess is wrong, anything written into the registers must be cleared, and the pipeline must be started again with the correct instruction. Some methods of branch prediction depend on stereotypical behavior. Branches pointing backward are taken about 90% of the time since backward-pointing branches are often found at the bottom of loops. On the other hand, branches pointing forward are only taken approximately 50% of the time. Thus, it would be logical for processors to always follow the branch when it points backward, but not when it points forward. Other methods of branch prediction are less static: processors that use dynamic prediction keep a history for each branch and use it to predict future branches. These processors are correct in their predictions 90% of the time.

d) ***Refer to Question No. 5(a) of Long Answer Type Questions.***