UNIT –IV
PROCESSOR

INSTRUCTION EXECUTION

Steps in Instruction Execution by CPU:

Six steps are involved in execution of an instruction by CPU. However, not all of them are required for all instructions.

1. Fetch instruction
2. Decode information
3. Perform ALU operation
4. Access memory
5. Update register file
6. Update the Program Counter (PC)

Step 1: Fetch instruction

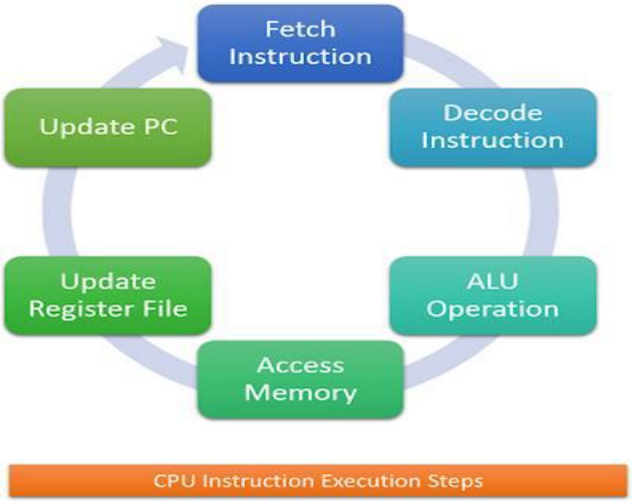
Execution cycle starts with fetching instruction from main memory. The instruction at the current program counter (PC) will be fetched and will be stored in instruction register (IR).

Step 2: Decode instruction

During this cycle the encoded instruction present in the IR (instruction register) is interpreted by the decoder.

Step 3: Perform ALU operation

ALU (Arithmetic Logic Unit) is where two operands in the instruction will be operated on given operator in the instructions. Such as, if the instruction was to add two numbers, then here the addition will happen. ALU take two values and output one, the result of the operation.



Step 4: Access memory

There are only two kind of instructions that access memory: LOAD and STORE. LOAD copies a value from memory to a register and STORE copies a register value to memory. Any other instruction skips this step.

Step 5: Update Register File

In this step, the output/result of the ALU is written back to the register file to update the register file. The result could also be due to a LOAD from memory. Some instructions don't have results to store. For example, BRANCH and JUMP instructions do not have any results to store.

Step 6: Update the PC (Program Counter)

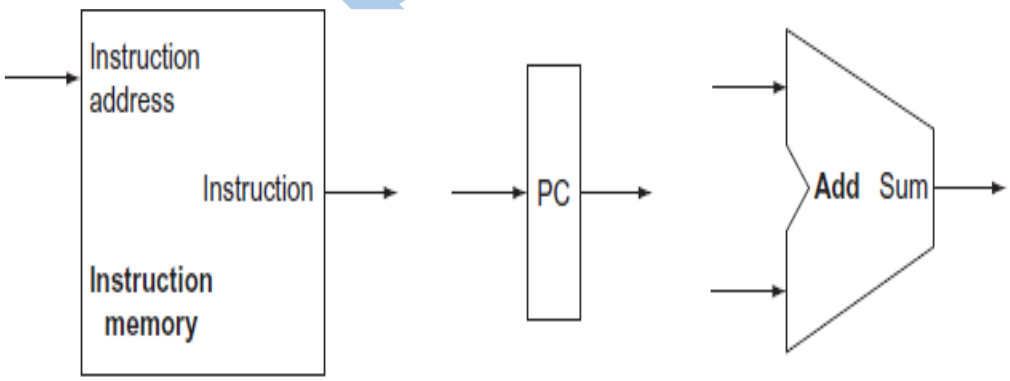
Ultimately, at the end of the execution of the current instruction, we need to update the program counter (PC) to the address of the next instruction, so that we can go back to step 1 where the CPU will fetch instruction. However, the program counter might need to be set to other memory address than the next one if the instruction was BRANCH or JUMP

BUILDING A DATAPATH

Datapath:

It is a processor components that are used to perform arithmetic operations and elements that holds the data.

Main elements of data path are Instruction memory, Program Counter (PC), ALU adder.



a. Instruction memory b. Program counter c. Adder

Building a data path

MIPS processor data path can be built incrementally by only considering the subsets of instruction.

Types of Elements in the Datapath

State element:

· A memory element, i.e., it contains a state

· E.g., program counter, instruction memory Combinational element:

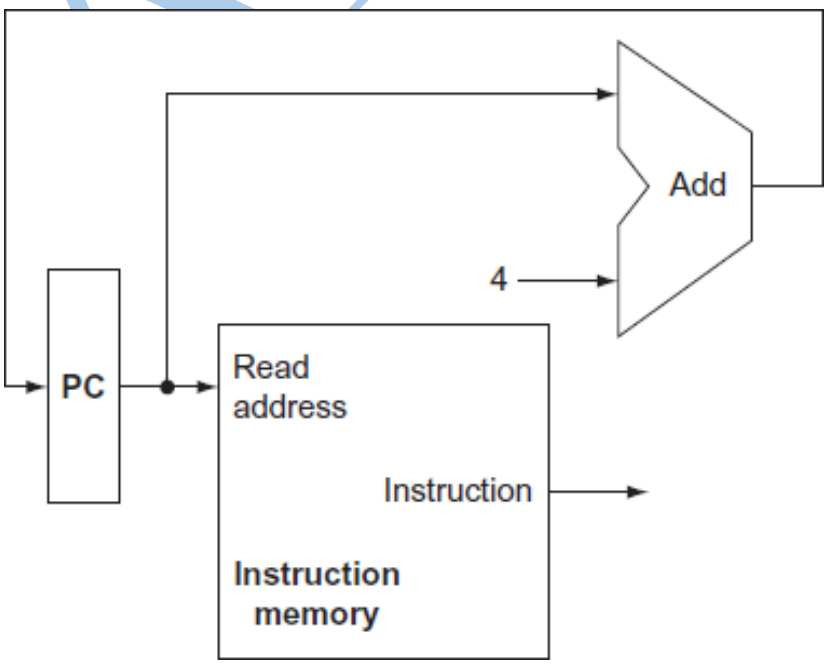
· Elements that operate on values
· Eg adder ALU E.g. adder, ALU

Elements required by the different classes of instructions
· Arithmetic and logical instructions
· Data transfer instructions
· Branch instructions

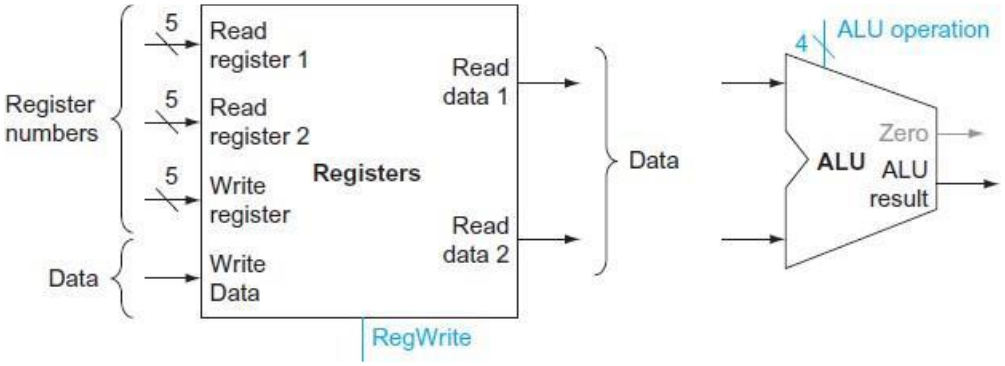
R-Format ALU Instructions
· E.g., add $t1, $t2, $t3
· Perform arithmetic/logical operation
· Read two register operands and write register result Register file:

· A collection of the registers
· Any register can be read or written by specifying the number of the register
· Contains the register state of the computer ALU
· Takes two 32 bit input and produces a 32 bit output
· Also, sets one-bit signal if the results is 0

· The operation done by ALU is controlled by a 4 bit control signal input. This is set according to the instruction .
Portion of data path used for fetching instructions and incrementing the program counter



Register type instructions ADD $t1, $t2, $t3



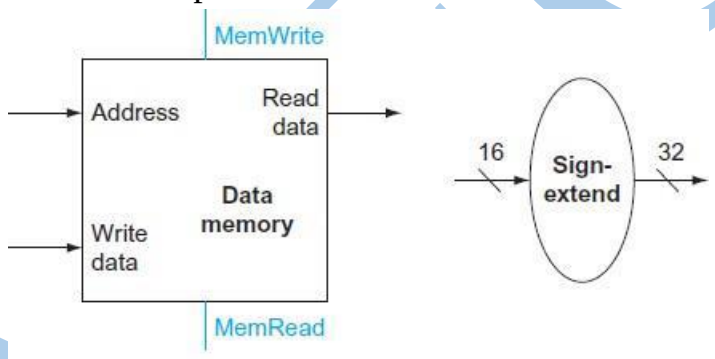
a. Registers b. ALU

Use of sign extension unit

It is used to convert the 16 bit constant input into 32 bit constant output. Addi $R1, $R2, 20

Here 20 is the 16 bit constant.

Sign extension unit is used to convert this 16 bit constant into 32 bit output and given to the ALU unit as a input



a. Data memory unit b. Sign extension unit

Use of shift left 2

Shift left 2 operation is used in conditional branch instruction Beq $t0, $t1, 250

It is used to find the exact byte address from the branch address specified in word (here 250 is word no.)

Control branch : PC = PC + 250

If we perform the shift left 2 operation on 250 we can obtain

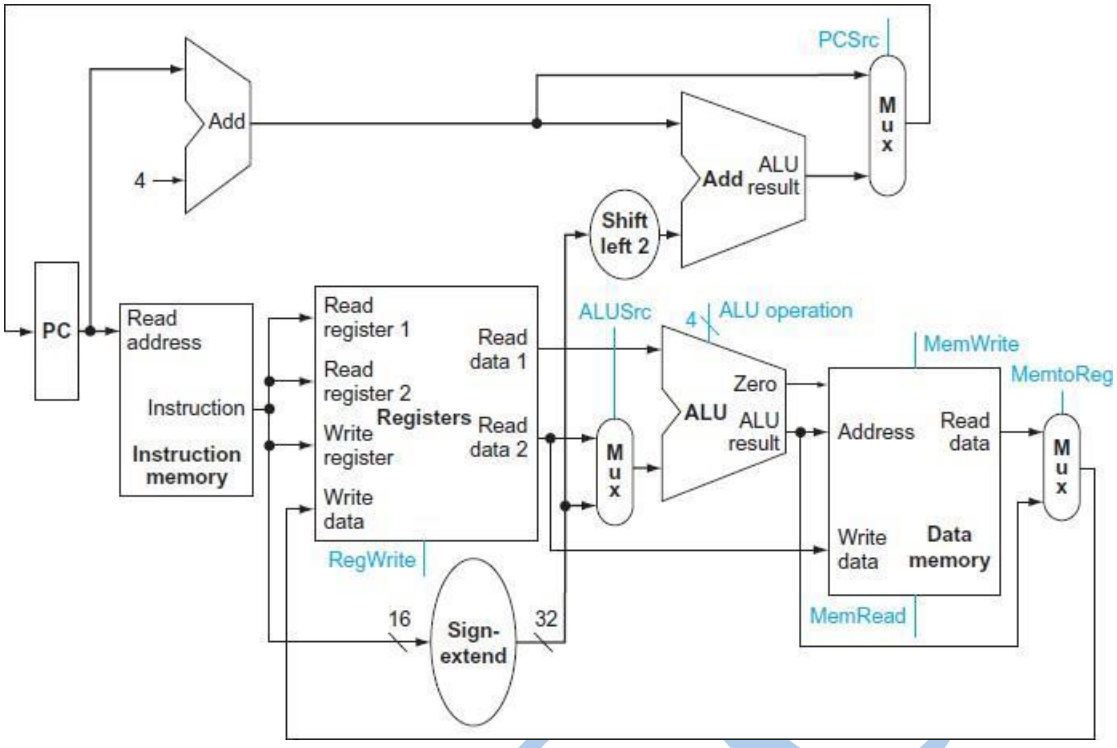
1000.

The control branch address is, PC = PC +1000

Creating a single data path

Here all the operations are included in single data path.

Operations such as R-Type, I-Type, J-type are included in this datapath.

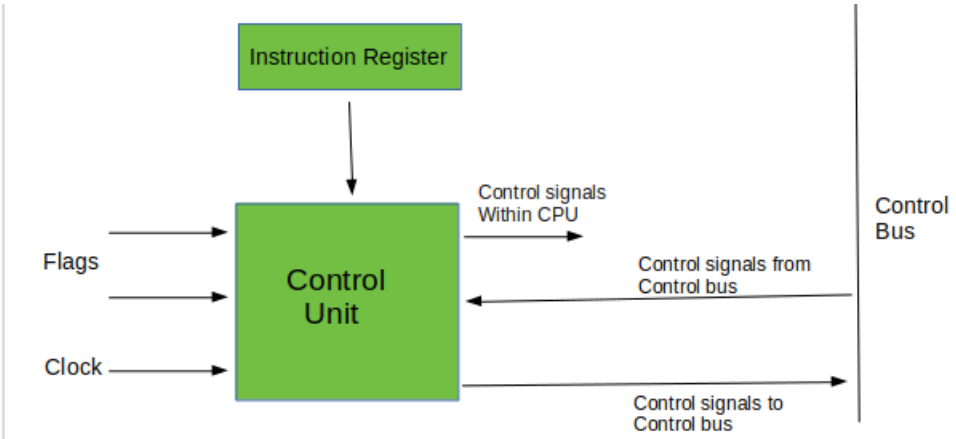


DESIGNING OF CONTROL UNIT

Control Unit is the part of the computer's central processing unit (CPU), which directs the operation of the processor. It was included as part of the Von Neumann Architecture by John von Neumann. It is the responsibility of the Control Unit to tell the computer's memory, arithmetic/logic unit and input and output devices how to respond to the instructions that have been sent to the processor. It fetches internal instructions of the programs from the main memory to the processor instruction register, and based on this register contents, the control unit generates a control signal that supervises the execution of these instructions.

A control unit works by receiving input information to which it converts into control signals, which are then sent to the central processor. The computer's processor then tells the attached hardware what operations to perform. The functions that a control unit performs are dependent on the type of CPU because the architecture of CPU varies from manufacturer to manufacturer. Examples of devices that require a CU are:

- Control Processing Units(CPUs)
- Graphics Processing Units(GPUs)



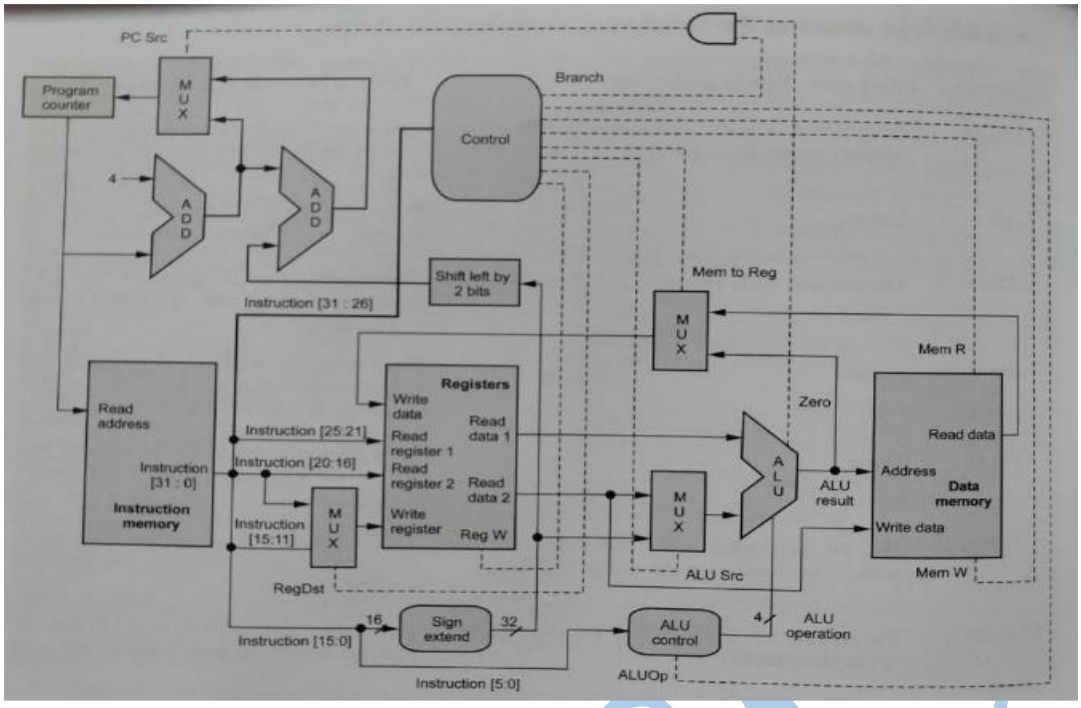
Block Diagram of the Control Unit

Functions of the Control Unit –

1. It coordinates the sequence of data movements into, out of, and between a processor's many sub-units.
2. It interprets instructions.
3. It controls data flow inside the processor.
4. It receives external instructions or commands to which it converts to sequence of control signals.
5. It controls many execution units(i.e. ALU, data buffers and registers) contained within a CPU.
6. It also handles multiple tasks, such as fetching, decoding, execution handling and storing results.

Truth table for ouptut control lines and ALUOp

Instruction	Input Op [5 : 0]						Reg Dst	ALU scr	Mem toReg	Reg W	Mem R	Mem W	Branch	ALU Op1	ALU Op2
	5	4	3	2	1	0									
R-format	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0
lw	1	0	0	0	1	1	0	1	1	1	1	0	0	0	0
sw	1	0	1	0	1	1	x	1	x	0	0	1	0	0	0
beq	0	0	0	1	0	0	x	0	x	0	0	0	1	0	1



Simple data path with the control uni

Types of Control Unit – There are two types of control units: Hardwired control unit and Micro programmable control unit.

HARDWIRED CONTROL METHOD

It is a hardware based method to design the control unit.

o A Hard-wired Control consists of two decoders, a sequence counter, and a number of logic gates.

o An instruction fetched from the memory unit is placed in the instruction register (IR).

o The component of an instruction register includes; I bit, the operation code, and bits 0 through 11.

o The operation code in bits 12 through 14 are coded with a 3 x 8 decoder.

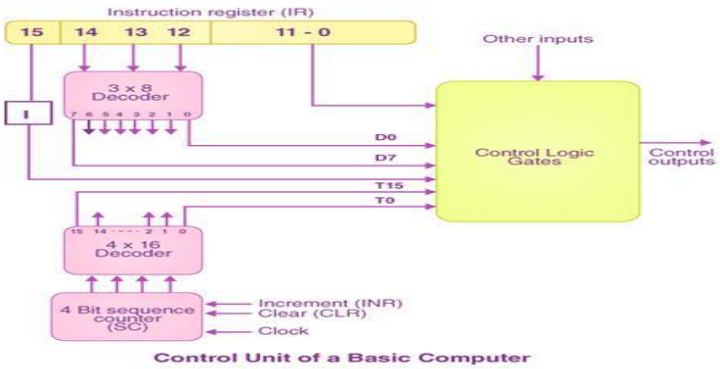
o The outputs of the decoder are designated by the symbols D0 through D7.

o The operation code at bit 15 is transferred to a flip-flop designated by the symbol I.

o The operation codes from Bits 0 through 11 are applied to the control logic gates.

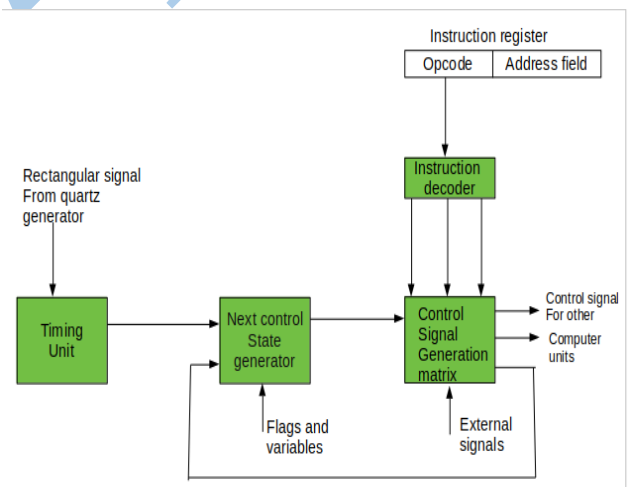
o The Sequence counter (SC) can count in binary from 0 through 15

In the Hardwired control unit, the control signals that are important for instruction execution control are generated by specially designed hardware logical circuits, in which we can not modify the signal generation method without physical change of the circuit structure. The operation code of an instruction contains the basic data for control signal generation. In the instruction decoder, the operation code is decoded. The instruction decoder constitutes a set of many decoders that decode different fields of the instruction opcode.



Control Unit of a Basic Computer

As a result, few output lines going out from the instruction decoder obtains active signal values. These output lines are connected to the inputs of the matrix that generates control signals for execution units of the computer. This matrix implements logical combinations of the decoded signals from the instruction opcode with the outputs from the matrix that generates signals representing consecutive control unit states and with signals coming from the outside of the processor, e.g. interrupt signals. The matrices are built in a similar way as a programmable logic arrays.



Block diagram of a hardwired control unit of a computer

Control signals for an instruction execution have to be generated not in a single time point but during the entire time interval that corresponds to the instruction execution cycle. Following the structure of this cycle, the suitable sequence of internal states is organized in the control unit.

A number of signals generated by the control signal generator matrix are sent back to inputs of the next control state generator matrix. This matrix combines these signals with the timing signals, which are generated by the timing unit based on the rectangular patterns usually supplied by the quartz generator. Whena new instruction arrives at the control unit, the control units is in the initial state of new instruction fetching. Instruction decoding allows the control unit enters the first state relating execution of the new instruction, which lasts as long as the timing signals and other input signals as flags and state information of the computer remain unaltered. A change of any of the earlier mentioned signals stimulates the change of the control unit state.

This causes that a new respective input is generated for the control signal generator matrix. When an external signal appears, (e.g. an interrupt) the control unit takes entry into a next control state that is the state concerned with the reaction to this external signal (e.g. interrupt processing). The values of flags and state variables of the computer are used to select suitable states for the instruction execution cycle.

The last states in the cycle are control states that commence fetching the next instruction of the program: sending the program counter content to the main memory address buffer register and next, reading the instruction word to the instruction register of computer. When the ongoing instruction is the stop instruction that ends program execution, the control unit enters an operating system state, in which it waits for a next user directive.

Pros(advantage) of Hardwired Control Unit

 Hardwired Control Unit is quick due to the usage of combinational circuits to generate signals.

 The amount of delay that can occur in the creation of control signals is dependent on the number of gates.

 It can be tweaked to get the fastest mode of operation.

 Quicker than a micro-programmed control unit.

Cons(disadvantage) of Hardwired Control Unit

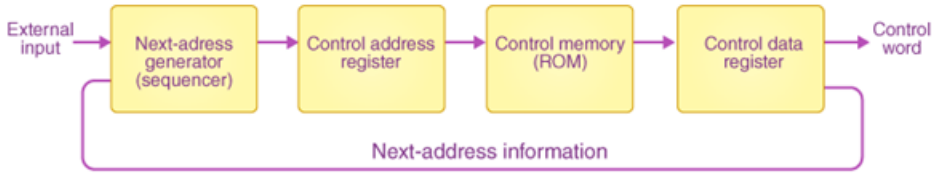
 As it require additional control signals to be created, the design becomes more complex (need for more encoders or decoders).

 Changes to control signals are challenging since they necessitate rearranging wires in the hardware circuit.

 It's difficult and time-consuming to add a new feature.

 It's difficult to evaluate and fix flaws in the initial design. It consists of micro program stored in control memory called ROM.

Micro program consists of large number of micro instructions. When executing a micro instruction it will generate a appropriate control signal.



Microprogrammed control unit of a Basic Computer

Next address generator examine the instruction within Instruction Register(IR) and find the address of micro instruction within micro program to create the corresponding control signal.

Send the micro instruction address to control address register. Now control address register contain the address of microinstruction.

Then the appropriate microinstruction is pick from the control memory (ROM). (Control memory contains the micro program)

The micro instruction is then shifted to control data register.

The data processor executes the microinstruction to produce the control signal.

While executing the micro instruction, next address generator generate the address of the micro instruction needed to execute next.

The address of micro instruction stored control address register(micro instruction address register) is decoded then it given as input to control memory(control store)

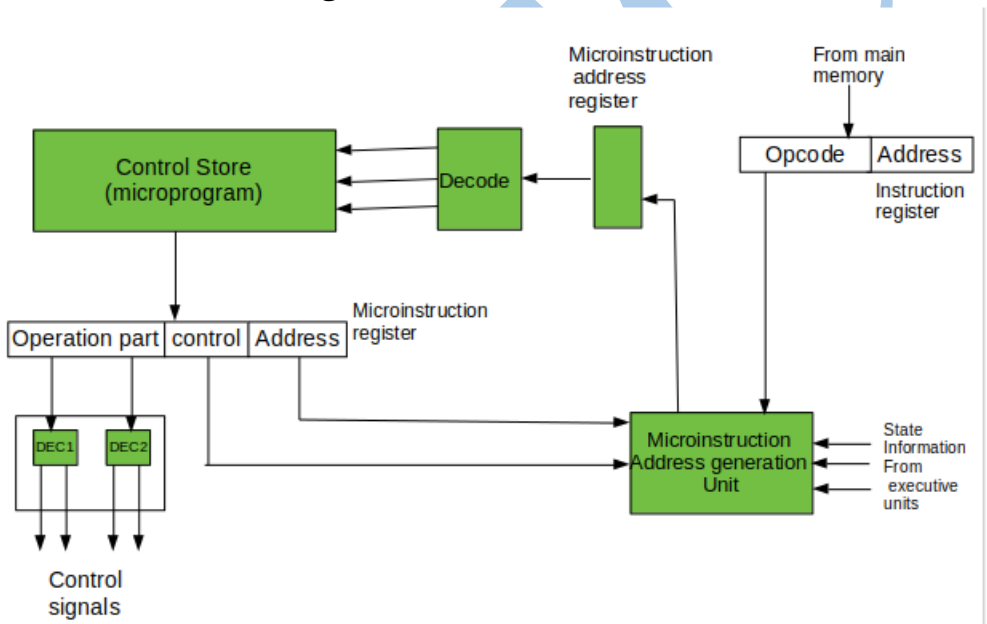
Micro instructions operation part is decoded and produce the control signals.

Micro instructions control part and address part is useful to generate the next micro instruction address and provide this information to next address generator.

The fundamental difference between these unit structures and the structure of the hardwired control unit is the existence of the control store that is used for storing words containing encoded control signals mandatory for instruction execution.

In microprogrammed control units, subsequent instruction words are fetched into the instruction register in a normal way. However, the operation code of each instruction is not directly decoded to enable immediate control signal generation but it comprises the initial address of a microprogram contained in the control store.

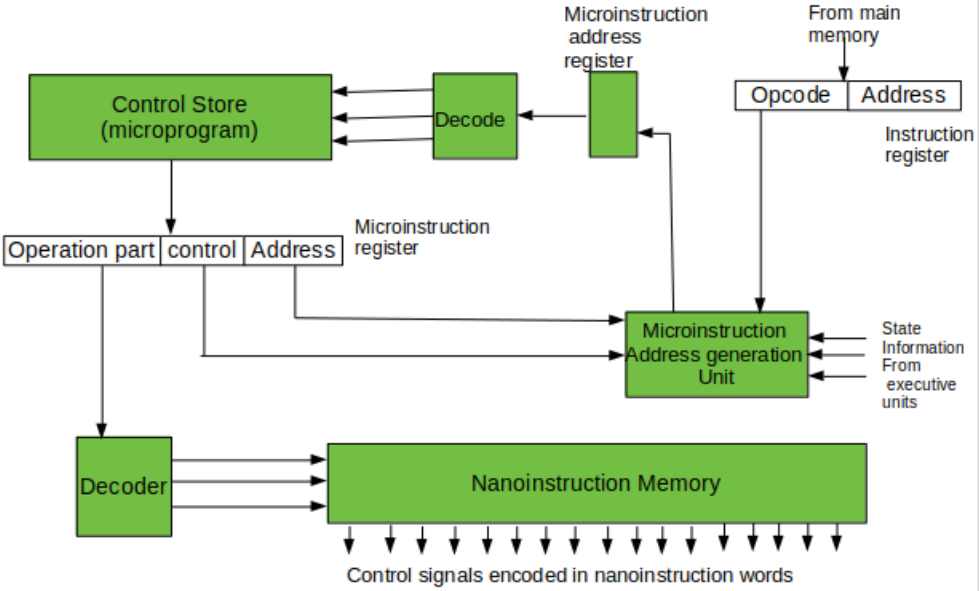
 With a single-level control store: In this, the instruction opcode from the instruction register is sent to the control store address register. Based on this address, the first microinstruction of a microprogram that interprets execution of this instruction is read to the microinstruction register. This microinstruction contains in its operation part encoded control signals, normally as few bit fields. In a set microinstruction field decoders, the fields are decoded. The microinstruction also contains the address of the next microinstruction of the given instruction microprogram and a control field used to control activities of the microinstruction address generator.



Microprogrammed control unit with a single level control store

The last mentioned field decides the addressing mode (addressing operation) to be applied to the address embedded in the ongoing microinstruction. In microinstructions along with conditional addressing mode, this address is refined by using the processor condition flags that represent the status of computations in the current program. The last microinstruction in the instruction of the given microprogram is the microinstruction that fetches the next instruction from the main memory to the instruction register

 With a two-level control store: In this, in a control unit with a two-level control store, besides the control memory for microinstructions, a nano-instruction memory is included. In such a control unit, microinstructions do not contain encoded control signals. The operation part of microinstructions contains the address of the word in the nano-instruction memory, which contains encoded control signals. The nano-instruction memory contains all combinations of control signals that appear in microprograms that interpret the complete instruction set of a given computer, written once in the form of nano-instructions.



Microprogrammed control unit with a two-level control store

In this way, unnecessary storing of the same operation parts of microinstructions is avoided. In this case, microinstruction word can be much shorter than with the single level control store. It gives a much smaller size in bits of the microinstruction memory and, as a result, a much smaller size of the entire control memory. The microinstruction memory contains the control for selection of consecutive microinstructions, while those control signals are generated at the basis of nano-instructions. In nano-instructions, control signals are frequently encoded using 1 bit/ 1 signal method that eliminates decoding.

Micro programmed Control Unit Pros(advantage)

 It allows for a more methodical control unit design.

 It's easier to troubleshoot and modify.

 It can keep the control function's fundamental structure.

 It can make the control unit's design easier. As a result, it is less expensive and less prone to errors or glitches.

 It has the ability to design in a methodical and ordered manner.

 It is used to control software-based functions rather than hardware-based functions.

 It's more adaptable.

 It is used to do complex functions with ease.

Microprogrammed Control Unit Cons(disadvantage)

 Adaptability comes at a higher price.

 It is comparatively slower than a control unit that is hardwired.

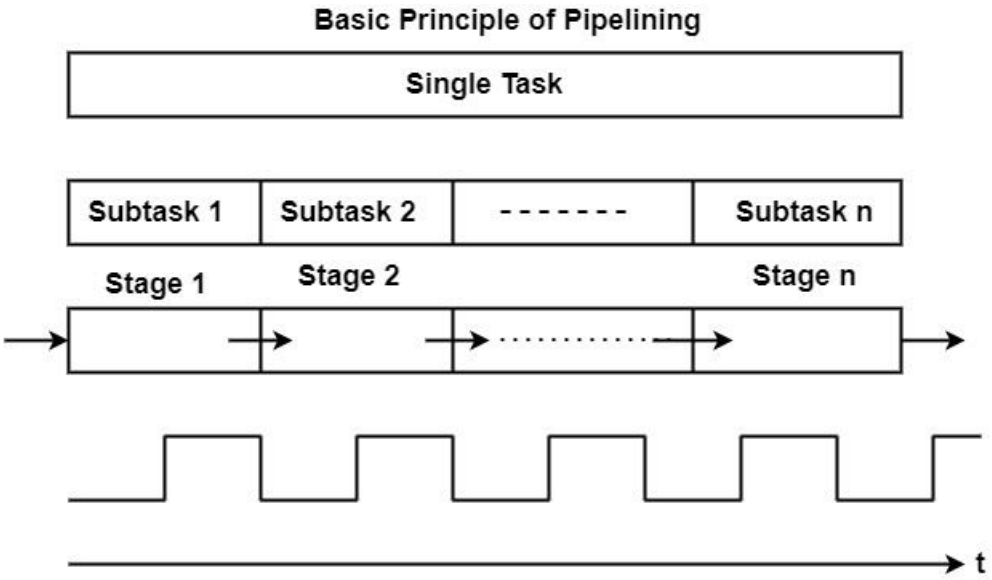
Hardwired control	**Micro programmed control**
i. It is a hardware based method to design control unit	It is programming method to design control unit
ii. Here control logic circuits generate the control signals	Here micro instructions generate the control signals
iii. It is difficult one to modify the method of generating control signals	It is easier one to modify
iv. Higher speed than micro programmed control method	It is slower than hardwired control
v. Costlier than micro program method	cheaper than hardwired method

PIPELINING

Pipelining defines the temporal overlapping of processing. Pipelines are emptiness greater than assembly lines in computing that can be used either for instruction processing or, in a more general method, for executing any complex operations. It can be used efficiently only for a sequence of the same task, much similar to assembly lines.

A basic pipeline processes a sequence of tasks, including instructions, as per the following principle of operation −

Each task is subdivided into multiple successive subtasks as shown in the figure. For instance, the execution of register-register instructions can be broken down into instruction fetch, decode, execute, and writeback.



A pipeline phase related to each subtask executes the needed operations.

A similar amount of time is accessible in each stage for implementing the needed subtask.

All pipeline stages work just as an assembly line that is, receiving their input generally from the previous stage and transferring their output to the next stage.

Finally, it can consider the basic pipeline operates clocked, in other words synchronously. This defines that each stage gets a new input at the beginning of the clock cycle, each stage has a single clock cycle available for implementing the needed operations, and each stage produces the result to the next stage by the starting of the subsequent clock cycle.

Five phases of instruction execution

1. Instruction Fetch
2. Instruction decode
3. Instruction execution
4. Memory access
5. Write back

Pipelining take this instruction execution phases into pipelining stages.

Pipelining stages

i. Instruction Fetch(IF)

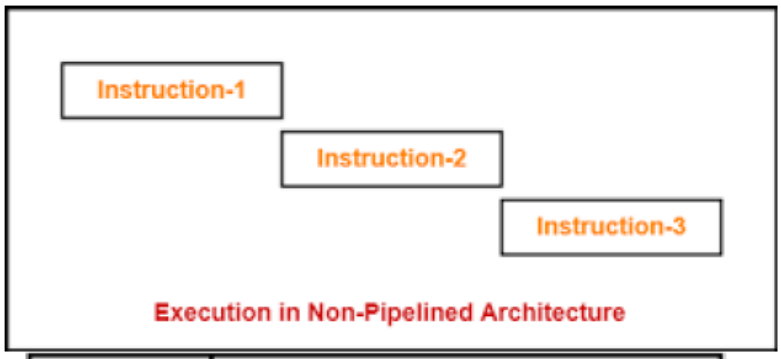
ii. Instruction Decode (ID)

iii. Instruction Execution (ALU)

iv. Memory Access (MA)

v. Write Back(WB)

Pipelining is used to improve the program execution speed. It never leave the processor components in idle position.
Each stage in pipeline is executed in one clock cycle of CPU.



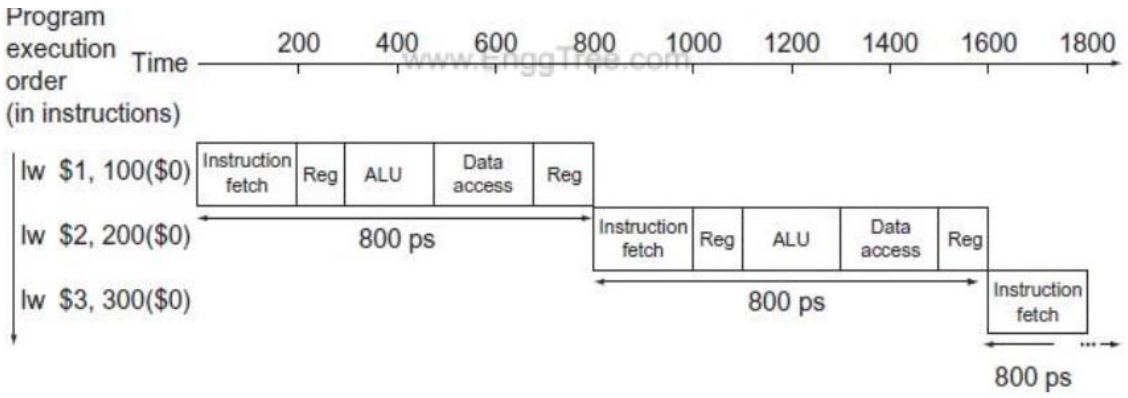
Instr No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

$$\text{Time between instructions}_{\text{pipelined}} = \frac{\text{Time between instruction}_{\text{nonpipelined}}}{\text{Number of pipe stages}}$$

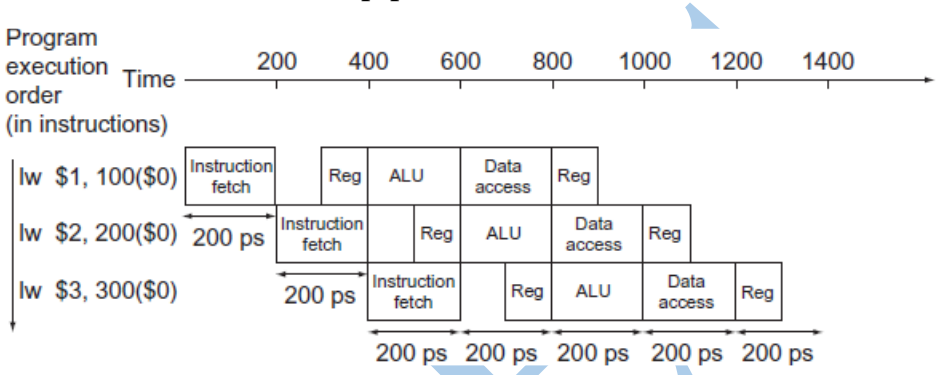
Exact time required to finish each stage of pipeline in MIPS processor is given as follows

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (`lw`)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (`sw`)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (`add`, `sub`, `AND`, `OR`, `slt`)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (`beq`)	200 ps	100 ps	200 ps			500 ps

Following diagram shows time between instruction in non pipelined execution and pipelined execution.



Non pipelined execution



Types of pipelines

There are two types of pipelines in computer processing.

Instruction pipeline

The instruction pipeline represents the stages in which an instruction is moved through the various segments of the processor, starting from fetching and then buffering, decoding and executing. One segment reads instructions from the memory, while, simultaneously, previous instructions are executed in other segments. Since these processes happen in an overlapping manner, the throughput of the entire system increases. The pipeline's efficiency can be further increased by dividing the instruction cycle into equal-duration segments.

Pipeline processing can occur not only in the data stream but in the instruction stream as well.

Most of the digital computers with complex instructions require instruction pipeline to carry out operations like fetch, decode and execute instructions.

In general, the computer needs to process each instruction with the following sequence of steps.

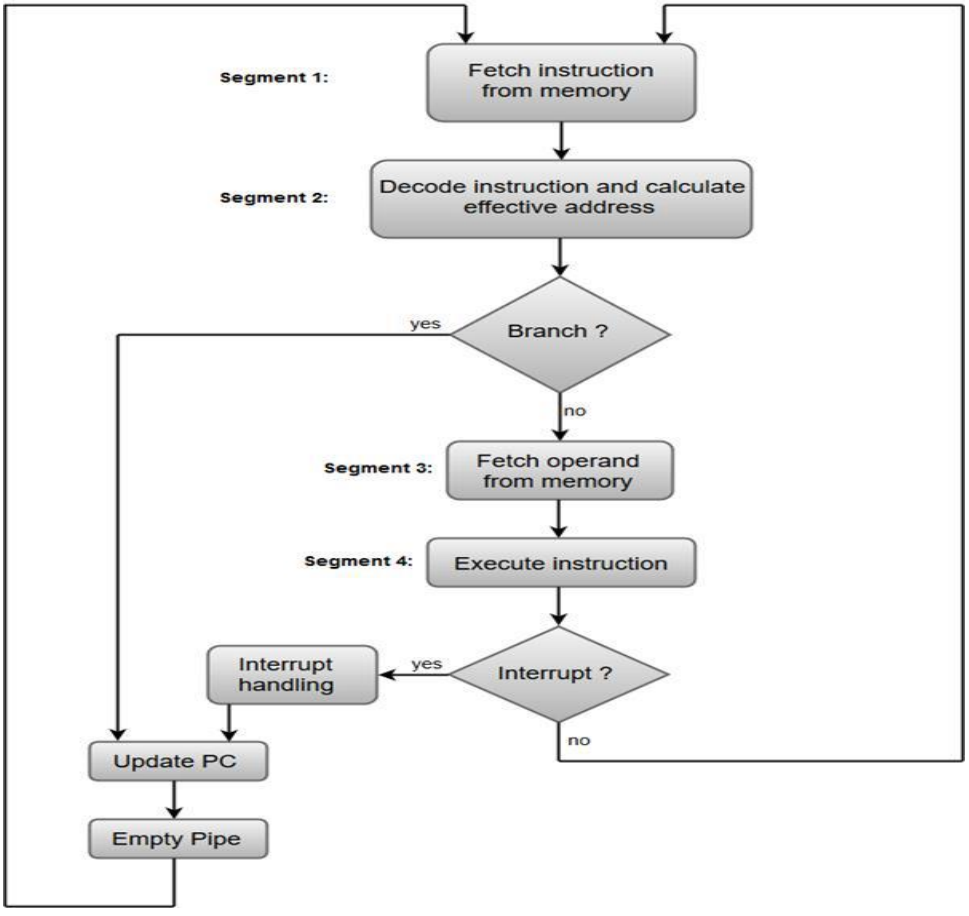
1. Fetch instruction from memory.
2. Decode the instruction.
3. Calculate the effective address.

4. Fetch the operands from memory.
5. Execute the instruction.
6. Store the result in the proper place.

Each step is executed in a particular segment, and there are times when different segments may take different times to operate on the incoming information. Moreover, there are times when two or more segments may require memory access at the same time, causing one segment to wait until another is finished with the memory. The organization of an instruction pipeline will be more efficient if the instruction cycle is divided into segments of equal duration. One of the most common examples of this type of organization is a **Four-segment instruction pipeline**

A **four-segment instruction** pipeline combines two or more different segments and makes it as a single one. For instance, the decoding of the instruction can be combined with the calculation of the effective address into one segment.

The following block diagram shows a typical example of a four-segment instruction pipeline. The instruction cycle is completed in four segments



Segment 1:
The instruction fetch segment can be implemented using first in, first out (FIFO) buffer.

Segment 2:
The instruction fetched from memory is decoded in the second segment, and eventually, the effective address is calculated in a separate arithmetic circuit.
Segment 3:
An operand from memory is fetched in the third segment.
Segment 4:
The instructions are finally executed in the last segment of the pipeline organization.

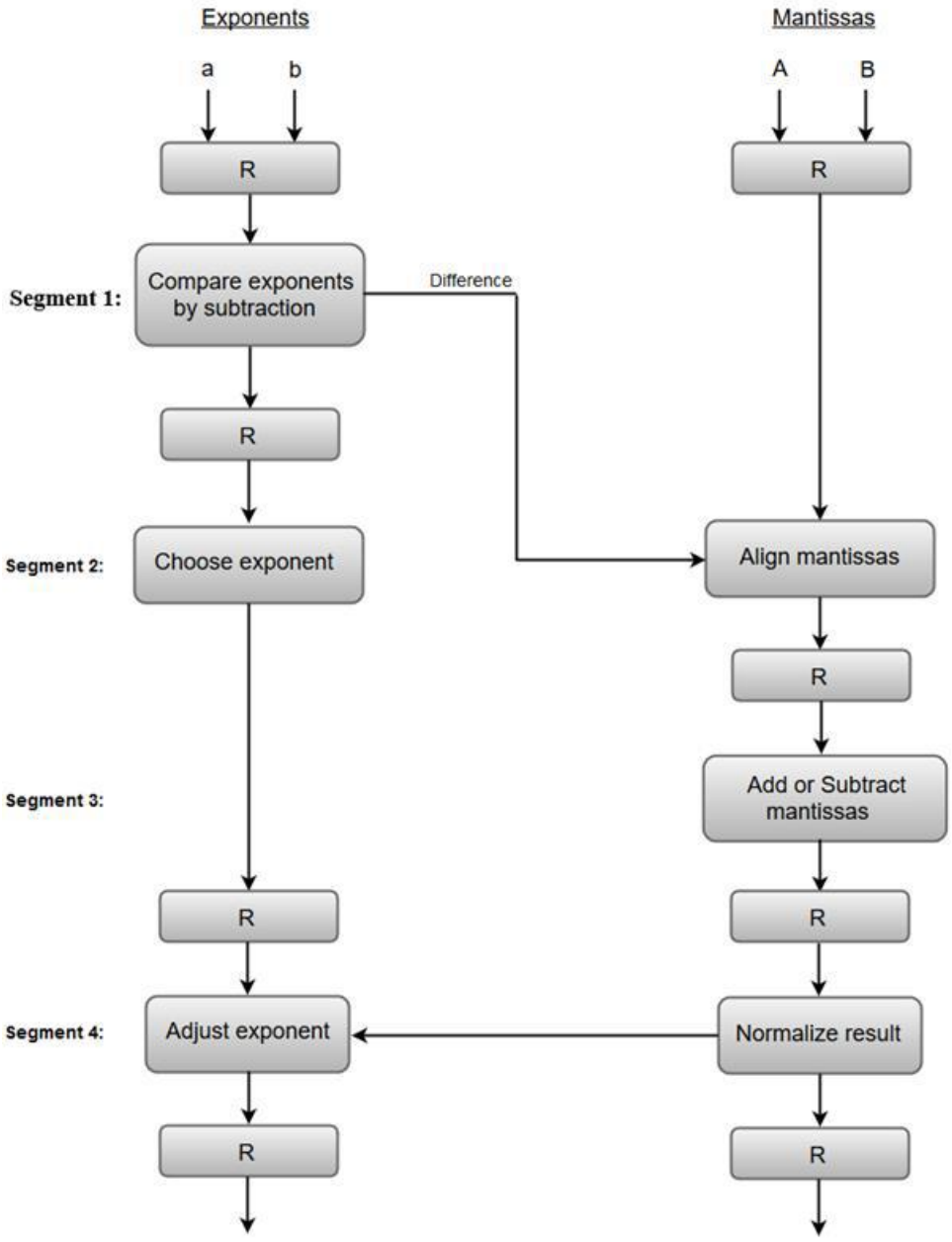
Arithmetic pipeline

Arithmetic Pipelines are mostly used in high-speed computers. They are used to implement floating-point operations, multiplication of fixed-point numbers, and similar computations encountered in scientific problems.
To understand the concepts of arithmetic pipeline in a more convenient way, let us consider an example of a pipeline unit for floating-point addition and subtraction.
The inputs to the floating-point adder pipeline are two normalized floating-point binary numbers defined as:

X = A * 2a = 0.9504 * 103
Y = B * 2b = 0.8200 * 102
Where **A** and **B** are two fractions that represent the mantissa and **a** and **b** are the exponents.
The combined operation of floating-point addition and subtraction is divided into four segments. Each segment contains the corresponding suboperation to be performed in the given pipeline. The suboperations that are shown in the four segments are:
1. Compare the exponents by subtraction.
2. Align the mantissas.
3. Add or subtract the mantissas.
4. Normalize the result.

The following block diagram represents the suboperations performed in each segment of the pipeline.

Pipeline organization for floating point addition and subtraction:



1. Compare exponents by subtraction:

The exponents are compared by subtracting them to determine their difference. The larger exponent is chosen as the exponent of the result.

The difference of the exponents, i.e., **3 - 2 = 1** determines how many times the mantissa associated with the smaller exponent must be shifted to the right.

2. Align the mantissas:

The mantissa associated with the smaller exponent is shifted according to the difference of exponents determined in segment one.

X = 0.9504 * 103

Y = 0.08200 * 103

3. Add mantissas:
The two mantissas are added in segment three.
Z = X + Y = 1.0324 * 103
4. Normalize the result:
After normalization, the result is written as:
Z = 0.1324 * 104

Advantages of Pipelining

 The cycle time of the processor is decreased. It can improve the instruction throughput. Pipelining doesn't lower the time it takes to do an instruction. Rather than, it can raise the multiple instructions that can be processed together ("at once") and lower the delay between completed instructions (known as 'throughput').

 If pipelining is used, the CPU Arithmetic logic unit can be designed quicker, but more complex.

 Pipelining increases execution over an un-pipelined core by an element of the multiple stages (considering the clock frequency also increases by a similar factor) and the code is optimal for pipeline execution.

 Pipelined CPUs frequently work at a higher clock frequency than the RAM clock frequency, (as of 2008 technologies, RAMs operate at a low frequency correlated to CPUs frequencies) increasing the computer's global implementation.

Disadvantages of Pipelining

Designing of the pipelined processor is complex. Instruction latency increases in pipelined processors. The throughput of a pipelined processor is difficult to predict. The longer the pipeline, worse the problem of hazard for branch instructions**.**

PIPELINE HAZARD

Dependencies between instructions in pipeline is called hazards
.Three types
i. Structural hazard

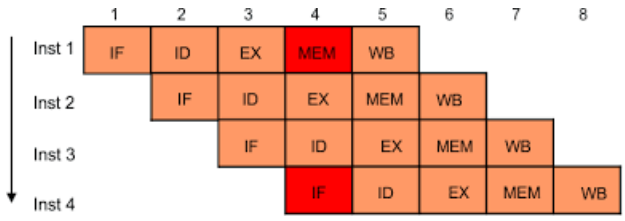
ii. Data hazard

iii. Control hazard

Structural hazard

Hardware dependencies between instructions in pipeline are called as structural hazard. Here hardware means memory, ALU, a register.

Example:



Here instruction1 and instruction 4 are use the memory at the same. So memory conflict will occur. At a time memory process only one work. Not able to do more than one work at a time (mutual exclusion property).

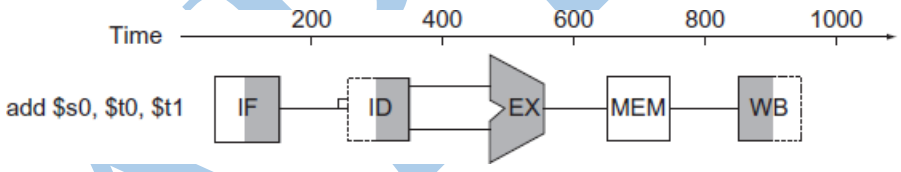
Pipeline stall

Delay in execution of instruction in pipeline is called pipeline stall. It is also called as bubble –nickname.

Data hazard

Data hazard occur when the pipeline must be stalled because one step must Wait for another to complete. Also called as pipeline data hazard. When a planned instruction can not execute in proper clock cycle because data that is need to

```
add   $s0, $t0, $t1
sub   $t2, $s0, $t3
```



execute the instruction is not yet available

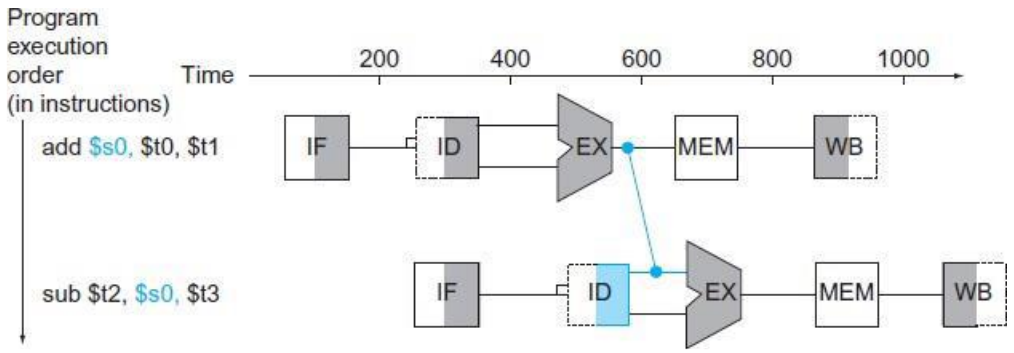
Consider following two instructions

Subtract operation needs the value of S0. But S0 value available write at WB stage

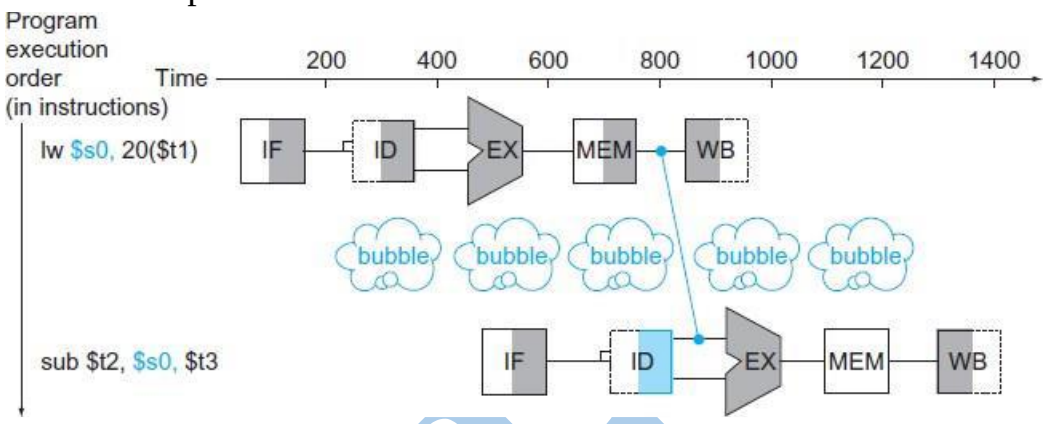
Forwarding

> The primary solution is based on the observation that we don't need to wait for the instruction to complete before trying to resolve the data hazard. For the code sequence above, as soon as the ALU creates the sum for the add, we can supply it as an input for the subtract. Adding extra hardware to retrieve the missing item early from the internal resources is called forwarding or bypassing.

Forwarding technique is useful to forward the results from internal hardware units.

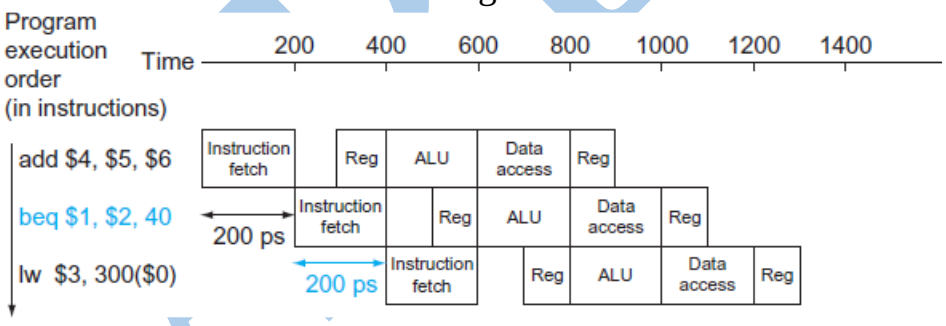


For another example



Control hazard

Control hazard arising from the need to make a decision based on the results of one instruction while others are executing.



In the above example , if $1 = $2 then go to the branch address PC=PC + 40x4
But here next load word instruction, lw $3, 300($0) is fetched from memory.
Suppose branch is taken then the pipeline wrongly fetch and execute the load word instruction

HANDLING DATA HAZARDS & CONTROL HAZARDS

Hazards: Prevent the next instruction in the instruction stream from executing during its designated clock cycle.

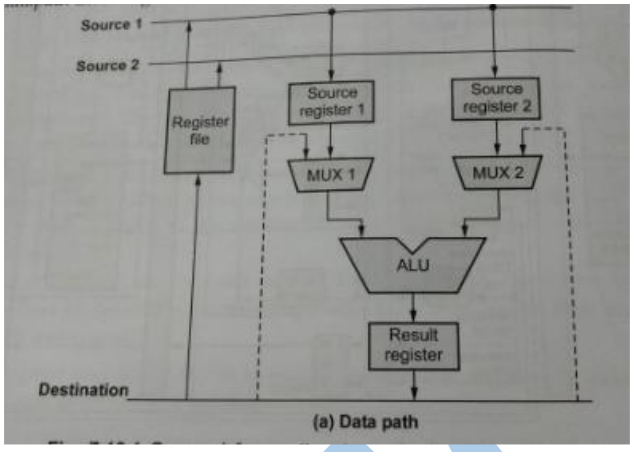
* Hazards reduce the performance from the ideal speedup gained by pipelining.

3 classes of hazards:

Ø Structural hazards: arise from resource conflicts when the hardware cannot support all possible combinations of instructions simultaneously in overlapped execution.

Ø Data hazards: arise when an instruction depends on the results of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline.

Ø Control hazards: arise from the pipelining of branches and other instructions that change the PC.



Performance of Pipelines with Stalls

* A stall causes the pipeline performance to degrade from the ideal performance.

Speedup from pipelining = [1/ (1+ pipeline stall cycles per instruction)] * Pipeline

$$\text{Speedup from pipelining} = \frac{1}{1+\text{pipeline stall cycles per instruction}} * \text{Pipeline depth}$$

Structural Hazards

* When a processor is pipelined, the overlapped execution of instructions requires pipelining of functional units and duplication of resources to allow all possible combinations of instructions in the pipeline.

* If some combination of instructions cannot be accommodated because of resource conflicts, the processor is said to have a structural hazard.

* Instances:

§ When functional unit is not fully pipelined, Then a sequence of instructions using that unpipelined unit cannot proceed at the rate of one per clock cycle.

§ when some resource has not been duplicated enough to allow all combinations of instructions in the pipeline to execute.

* To Resolve this hazard,

§ Stall the the pipeline for 1 clock cycle when the data memory access occurs. A stall is commonly called a pipeline bubble or just bubble, since it floats through the pipeline taking space but carrying no useful work.

Data Hazards

* A major effect of pipelining is to change the relative timing of instructions by overlapping their execution. This overlap introduces data and control hazards.
* Data hazards occur when the pipeline changes the order of read/write accesses to operands so that the order differs from the order seen by sequentially executing instructions on an unpipelined processor.

Minimizing Data Hazard Stalls by Forwarding

* The problem solved with a simple hardware technique called forwarding (also called bypassing and sometimes short-circuiting). Forwards works as:

§ The ALU result from both the EX/MEM and MEM/WB pipeline registers is always fed back to the ALU inputs.

§ If the forwarding hardware detects that the previous ALU operation has written the register corresponding to a source for the current ALU operation, control logic selects the forwarded result as the ALU input rather than the value read from the register file.

Data Hazards Requiring Stalls

* The load instruction has a delay or latency that cannot be eliminated by forwarding alone. Instead, we need to add hardware, called a pipeline interlock, to preserve the correct execution pattern.
* A pipeline interlock detects a hazard and stalls the pipeline until the hazard is cleared.
* This pipeline interlock introduces a stall or bubble. The CPI for the stalled instruction increases by the length of the stall.

Branch Hazards

* Control hazards can cause a greater performance loss for our MIPS pipeline . When a branch is executed, it may or may not change the PC to something other than its current value plus 4.
* If a branch changes the PC to its target address, it is a taken branch; if it falls through, it is not taken, or untaken.

Reducing Pipeline Branch Penalties

* Simplest scheme to handle branches is to freeze or flush the pipeline, holding or deleting any instructions after the branch until the branch destination is known.
* A higher-performance, and only slightly more complex, scheme is to treat every branch as not taken, simply allowing the hardware to continue as if the branch were not

executed. The complexity of this scheme arises from having to know when the state might be changed by an instruction and how to “back out” such a change.

* In simple five-stage pipeline, this predicted-not-taken or predicted untaken scheme is implemented by continuing to fetch instructions as if the branch were a normal instruction.

§ The pipeline looks as if nothing out of the ordinary is happening. § If the branch is taken, however, we need to turn the fetched instruction into a no-op and restart the fetch at the target address.

* An alternative scheme is to treat every branch as taken. As soon as the branch is decoded and the target address is computed, we assume the branch to be taken and begin fetching and executing at the target

Performance of Branch Schemes

Pipeline speedup = Pipeline depth / [1+ Branch frequency × Branch penalty]

$$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Branch frequency} \times \text{Branch penalty}}$$

The branch frequency and branch penalty can have a component from both unconditional and conditional branches.

APEC