UNIT – III
PROCESSOR AND CONTROL UNIT

Basic MIPS implementation – Building datapath – Control Implementation scheme – Pipelining – Pipelined datapath and control – Handling Data hazards & Control hazards – Exceptions.

1. IMPLEMENTATION OF MIPS

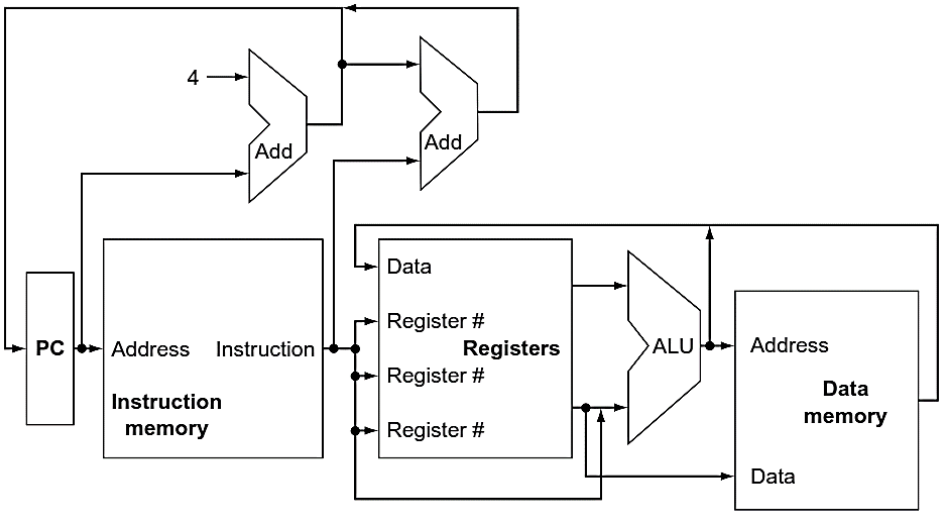
MIPS has three kinds of core instructions such as

1. Integer arithmetic logical instructions-add, sub, AND, OR and slt.
2. Memory reference instructions - load word (lw), store word (sw)
3. Branch instructions - jump (j) and branch equal (beq) .

To implement the above three types we have ***same method but independent of the exact class of instruction***. For every instruction, the first two steps are identical

1. Send the **PROGRAM COUNTER (PC)** to memory that contains the code and fetch the instruction from that memory.
2. Read one or two registers using field of the instruction to select the registers to read. .

Load word instruction need to read only one register but most other instruction require reading two registers. These two steps are common for all the instruction set. After these two steps, the actions required to complete the instruction depend on the instruction class.



MIPS instruction set has **simplicity and regularity** and it will simplify the implementation by making the execution of many of the instruction, classes similar.

For example, all instruction classes, except jump, use the **ARITHMETIC LOGICAL UNIT (ALU)** after reading the registers.

The memory reference instructions use the

- ALU for an address calculation
- Arithmetic logical instructions for the operation execution
- Branches for comparison

After using the ALU, the actions required to complete the task differs from various instruction classes.

- A ***memory reference instruction*** will need to access the memory either to read data for a load or write data for a store.
- An ***arithmetic logical or load instruction*** must write, the data from the ALU or memory back into a register. .

Branch instruction need to change the next instruction address based on the comparison, otherwise the PC should be incremented by 4 to get the address of the next instruction. All instructions start by using the program counter to supply the instruction address to the instruction memory.

After the instruction is fetched, the register operands used by an instruction are specified by fields of the instruction.

Once the register operands have been fetched, they can be operated to do the following tasks:

- To compute a memory address
- To compute an arithmetic result
- To compare for a branch

1.1. ARITHMETIC LOGICAL INSTRUCTION

If the instruction is arithmetic logical instruction then the result from the ALU must be written to a register.

1.2. LOAD OR STORE INSTRUCTION

If the operation is a load or store, the ALU result is used as an address to either store a value from the register or load a value from memory into the registers.

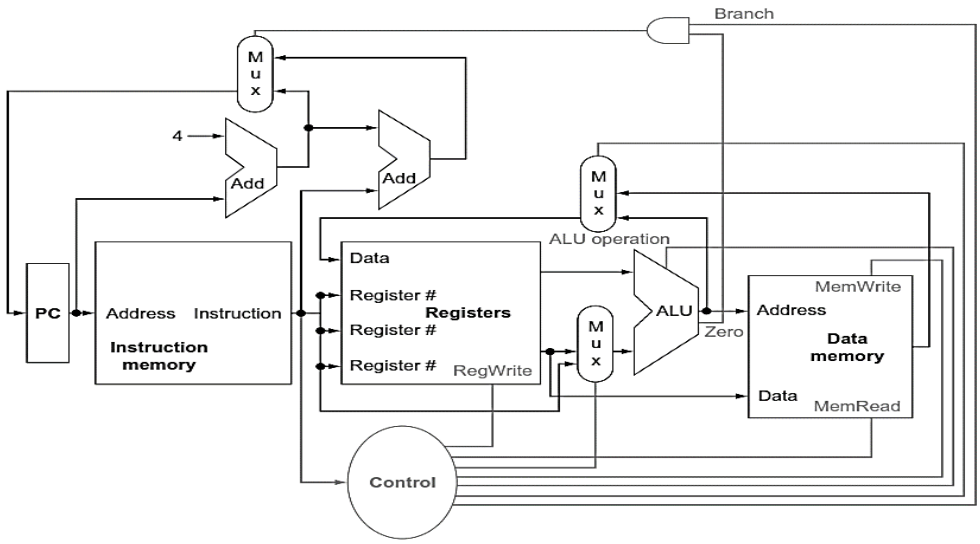
The result from the ALU or memory is written back into the register file.

1.3. BRANCH INSTRUCTIONS

Branch instruction require the use of the ALU output to determine the next instruction address, which comes either from the ALU or from an adder that increments the current PC by 4.

1.4. CONTROL UNIT

Control unit has the instruction as an input and it is used to determine how to set the control lines for the functional units and two of the multiplexers.



1.5. FUNCTIONS OF THE MULTIPLEXERS

The **TOP MULTIPLEXER** controls what value replaces the PC [PC + 4 or branch destination address].

The multiplexer is controlled by the gate that "AND"s together with the zero output of the ALU and control signal. Control signal indicates the branch instruction.

MIDDLE MULTIPLEXER is used to steer the output of the ALU or the output of the data memory for writing into a register file.

BOTTOM MULTIPLEXER is used to determine the second ALU input is from the registers or from the offset field of the instruction.

1.6. FUNCTION OF CONTROL LINE

Control lines are straight forward and determine the operation performed at the ALU.

ALU can perform following operations.

1. Data memory read
2. Data memory write
3. Write operation on register

Control line determines whether the ALU perform which operations among three mentioned operations. Control unit used to control actions taken for different instruction classes.

2. LOGICAL DESIGN CONVENTIONS

The data path elements in the MIPS implementation consists of two different types of logic elements such as

- Combinational element
- State element

COMBINATIONAL ELEMENT

Elements that operate on data values and elements that contain state. Combinational element means their outputs always depend only on the current inputs.

STATE ELEMENT

An element contains state if it has some internal storage. State element is a memory element such as a register or a memory. It is also called as sequential element.

State element has atleast two inputs and one output

The inputs are data values written into the element and the clock. Inputs are used to determine when the data value is written.

State element output produces a value written in an earlier clock cycle. .

The clock is used to determine when the state element should be written and state element can be read at any time.

Example: Flip flops and Registers

To design hardware designer must decide which element has to be used whether it is combinational or sequential.

2.1. CLOCKING METHODOLOGY

A clocking methodology defines when signals can be read and when they can be written.

Clocking methodology is designed to make hardware predictable;

It is important to specify the timing of reads and writes, because both read and write operation performed at same time means computer cannot predict none of the operation.

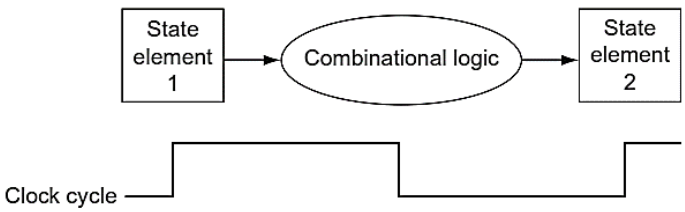
For example consider the edge triggered clocking methodology.

2.1.1. EDGE TRIGGERED CLOCKING

An edge triggered clocking methodology means any value stored in a sequential logic element are updated only on a clock edge.

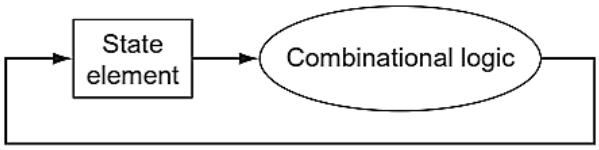
It is a quick transition from low to high or high to low.

In a synchronous digital system the clock determines when elements with the state will write values into internal storage.



All state elements including memory are assumed to be **positive edge triggered**.

So we can change the above figure as



Two state element and combinational logic operates in a single clock cycle.

All signals must propagate from state element 1 through the combinational logic and to state element 2 in one clock cycle.

The time required for the signals to reach state element 2 is called as **LENGTH OF THE CLOCK CYCLE**.

Control signal, is a signal used for' multiplexer selection for directing the operation of a functional unit.

Control signal contains information that is operated on by a functional unit.

Two logical terms are used to represent the signal level.

1. **Asserted term** used to indicate signal is logically high
2. **Deasserted term** used to indicate signal is logically low. .

Edge triggered methodology allows us to read the contents of a register.

After reading process was done send the value through some combinational logic and write the register in the same clock cycle.

For the 32 bit MIPS architecture all of these state and logic elements will have inputs and outputs as 32 bits wide. To obtain a 32 bit bus we have to combine two 16 bit buses.

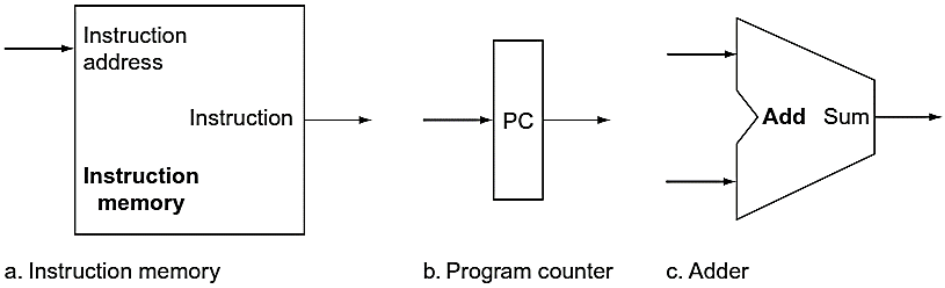
Any MIPS datapath using edge triggered writes must have more than one copy of the register file.

3. BUILDING A DATAPATH

In order to design a datapath we must list the major components required to execute each class of MIPS instructions. Components required to form a datapath is known as **DATA PATH ELEMENT**.

3.1. DATA PATH ELEMENT

It is a unit used to operate on or hold data within a processor. In the MIPS implementation, the datapath elements include the instruction and data memories, the register file, ALU and adders.



Above figure has two state elements

1. Instruction Memory
2. Program Counter.

The **instruction memory** needs ***only*** to provide ***read access*** because the datapath does not write instructions.

3.1.1. INSTRUCTION MEMORY

The instruction memory is called as combinational element because it will perform only read, the output at any time reflects the contents of the location specified by the address, input and no read control signal is needed.

3.1.2. PROGRAM COUNTER

The program counter is a 32 bit register that is written at the end of every clock and thus does not need a write control signal.

Program counter is a register containing the address of the instruction in the program being executed.

3.1.3. ADDER

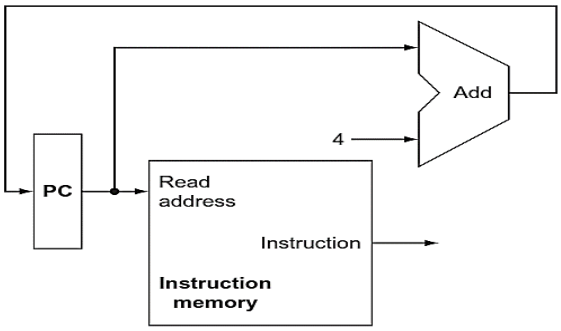
Adder is a combinational element used to add two 32 bit inputs and place the sum on its output.

3.2. FETCHING PHASE

To execute any instruction, we must start by fetching the instructions from memory.

To prepare for executing the next instruction, we have to increment the program counter.

The purpose of incrementing program counter is to point the next instruction by **INCREMENTING PC BY 4 BYTES.**



3.3. ARITHMETIC LOGICAL INSTRUCTIONS

It is also called as R-format instruction. To perform ALU operation these instructions **read two registers** and **writes** the result to **one register**.

It performs the operation on the contents of the registers and this instruction includes add, sub, AND, OR and sit.

Processors having 32 general purpose registers and some special purpose registers. General purpose and special purpose registers are stored in separate space of memory. 32 general purpose registers are stored in a structure called a **REGISTER FILE**.

3.3.1. REGISTER FILE

Register file is a state element that contains a set of registers that can be read and written by supplying a Register number to be accessed. Register file contains the register state of the computer.

3.4. R-TYPE INSTRUCTION OPERAND.

R-format instructions has three register operands to perform ALU operations. *Two registers to read data and one register to write data*

To read data word from the registers we have to specify the register number.

Register number specifies which data has to be read from which register present in the register file.

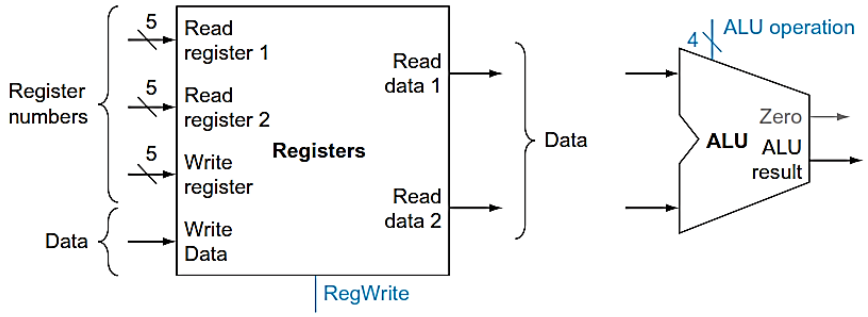
To write a data word into register we need two inputs:

1. One to specify the register number to be written.
2. One to supply the date to be written into the register.

Write operations are controlled by the write control signal and it must be asserted for a write to occur at the clock edge.

There are two elements we need to implement the R-format ALU operations such as

1. Register
2. ALU



a. Registers b. ALU

3.5. BRANCH INSTRUCTIONS

There are two kinds of branch instructions are available:

beq - branch equal

```
bnq -branch unequal
```

The **beq** instruction has three operands, in that

- Two operands are registers used to compare equality
- One operand is a16 bit offset used to compute the branch target address relative to the branch instruction address.

The beq instruction has the following form

```
beq   $tl,  $t2,  offset
```

To implement **beq** instruction we must find or compute the branch target address.

3.5.1. BRANCH TARGET ADDRESS

```
Branch target address = Sign extended offset field of the instruction + PC
```

It is an address specified in branch, which becomes the new program counter (PC) if the branch is taken.

If the operands are equal, the branch target address becomes the new PC and it is called as branch taken.

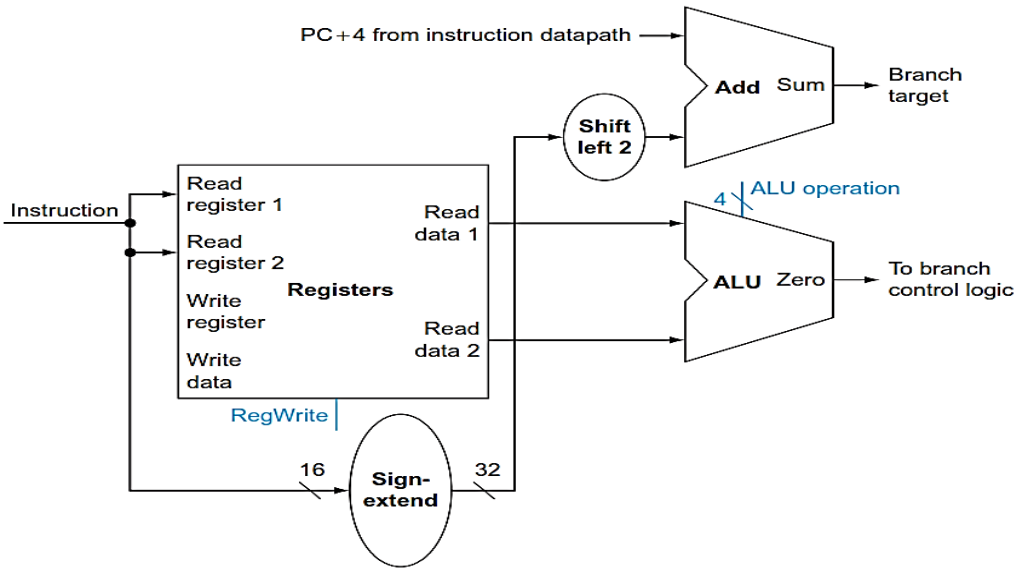
If the operands are not equal, the: incremented PC should replace the current PC and it is called as branch is not taken.

Branch data path will perform two kinds of operations.

1. Compute the branch target address
2. Compare the register contents

To compute the branch target address, the branch data path includes a sign extension unit.

To perform compare, we need to use the register file.



Adder circuit is used to compute the branch target and it is a sum of the incremented PC and the sign extended lower 16 bits of the instruction shifted left 2 units.

4. CREATING A SINGLE DATA PATH

To implement individual instruction classes we need the following datapath elements,

- Instruction Memory
- Program Counter
- ALU

By combining individual instruction class datapath components we can form a single data path and add the control to complete the implementation.

Single data path will execute all instructions in one clock cycle. It means no datapath resource can be used more than one per instruction. In single datapath if any element needs more than one clock cycle then that element must be duplicated.

4.1. SHARING THE DATAPATH

To share a datapath element between two different instruction classes we need to allow multiple connections to the input of an element. To provide multiple connections we need to use multiplexer and control signal to select among the multiple inputs.

Consider two different instruction classes are

[1] Arithmetic logical instructions (or) R-type
[2] Memory instructions

4.2. R-TYPE VS. MEMORY INSTRUCTION

R-type instruction	Memory instruction
It gets two operands from registers to perform ALU operation.	It gets one operand from register and another operand from sign extended 16 bit offset field from the instruction to do address calculation.
ALU result are stored in the destination register.	ALU result are stored using *load*.

For these two different kinds of instruction classes we need to make single datapath.

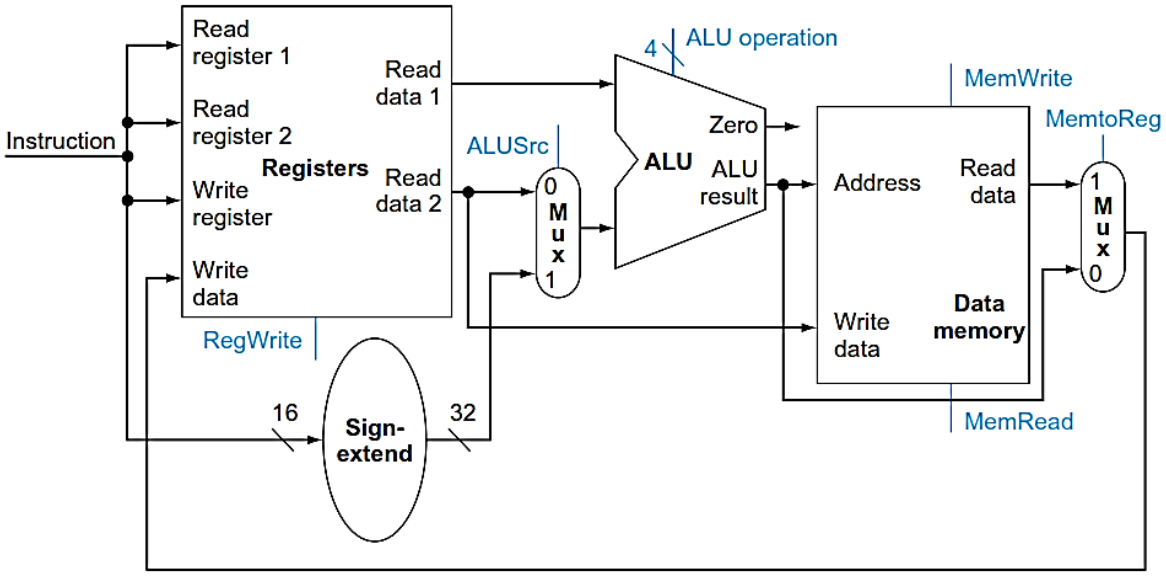
It can be obtained by using single register file, single ALU to handle both types of instructions and multiplexers.

To create a datapath with only single register file and single ALU we need to provide two different sources for the second input of the ALU.

Because both instructions has first operand as register and second operand is different.

As like input, two instructions has different format to store result so we need to support two different sources for the data stored into the register file.

For that we need ***two multiplexers***, one is placed at the ALU input and another is placed at the data input to the register file.



Datapath for the Memory Instructions and the R-Type Instructions

We can combine all the simple datapath components to make core MIPS architecture. It can be obtained by adding datapath for instruction fetch, datapath for R type and memory instructions and data path for branches, as shown in the figure

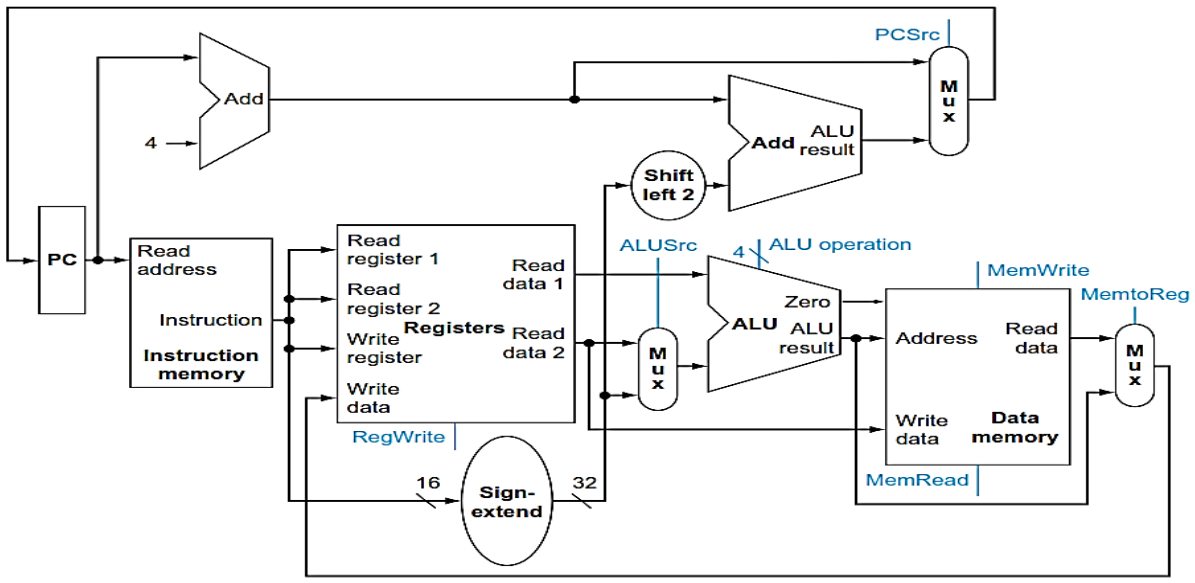
The branch instruction uses the main ALU for comparison of the register operands. So we need to use adder circuit for data path components of branch instruction.

An additional multiplexer is required to select either the sequentially following instruction address ***(PC + 4)*** or the ***branch target address*** to be written into the PC.

4.3. CONTROL UNIT

To complete simple datapath we must add the control unit also. Control unit must be able to take inputs and generate a write signal for

- Each state element,
- Selector control for each multiplexer
- ALU control.



Datapath for the Core MIPS Architecture

5. CONTROL IMPLEMENTATION SCHEME

MIPS ALU defines 6 following combinations of four control inputs:

Instruction class	Operation
0000	AND
0001	OR
0010	add
0110	subtract
0111	Set on less than
1100	NOR

Depending on the instruction class, the ALU need to perform one of these first five functions. NOR operation is needed for other parts of MIPS instruction set

For load word and store word instructions the ALU, has to compute the memory address by addition.

For R-type instructions, the ALU has to perform one of the five operations (AND, OR, add, subtract or set on less than) depending on the value of 6 bit function field in the low order bits of the instruction.

For branch equal (**beq**) the ALU has to perform subtraction.

5.1. ALUOP

We can generate the 4 bit ALU control input; using a small control unit called ALUOp. It has input function field of the instruction and a 2 bit control field

ALUOp indicates three kinds of operations.

[1] add (00) for loads and stores operations
[2] subtract (01) for branch equal

[3] Other operation determined by the function field (10)

The output of the ALU control unit is a 4 bit signal and it directly controls the ALU by generating one of the 4 bit combinations.

Instruction opcode	ALUOP	Instruction Operation	Funct field	Desired ALU action	ALU control input
LW	00	Load word	xxxxxx	add	0010
SW	00	Store word	xxxxxx	add	0010
Branch equal	01	Branch equal	xxxxxx	subtract	0110
R-type	10	Add	100000	add	0010
R-type	10	Subtract	100010	subtract	0110
R-type	1Q	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	Set on less than	101010	Set on less than	0111

Truth Table for ALUOp

Truth Table shows how the ALU control bits are set depends on the ALUOp control bits and the different function codes for the R-type instruction.

5.1.1. Determining ALU Action

Instruction opcode field determines the setting of the ALUOP bits. When the ALUOp is 00 or 01, the ALU action does not depend on the function code field. We do not care about the value of the function code and the function field is marked as XXXXXX (don't care term) for 00 and 01 values. When the ALUOp value is 10, then the function code is used to set the ALU control input.

5.1.2. Multilevel Decoding

The design uses multiple levels of decoding and it will provide the following functions.

- The main control unit generates the ALUOp bits.
- ALUOp bit is used as a input to the ALU control.
- That ALU control generates the actual signals to control the ALU unit.
- Multiple levels of controls can reduce the size of the main control unit.

Using several smaller control units may also increase the speed of the control unit. This type of optimizations are important, because the ***speed of the control unit is important to clock cycle time***.

5.2. Mapping 2-Bit ALUOp Field Into 6 Bit Function Field

There are several different ways to implement the mapping. From 2 bit ALUOP field and the 6 bit function field to the four ALU operation control bits.

There are 64 possible values available for function field of that only limited values are used more frequently. **The function field is used only when the ALUOP bits equal to 10.**

Once the truth table has been constructed it can be optimized and then turned into gates.

5.3. DESIGNING MAIN CONTROL UNIT

ALU control can be design using two ways

[1] Using the function code
[2] 2 bit signal used as a control inputs.

In order to design main control unit we have to **identify the fields** of an instruction and the **control lines**.

CONTROL LINES are needed for the data-path construction. Various instruction fields are connected together to form a single data path. We used three instruction classes and it is important to know the format of it. Because then only we can obtain a single data path by connecting different instruction classes.

Instruction format of R-type, load store and branch instructions are shown below.

Field	0	rs	rt	rd	shamt	funct
Bit positions	31:26	25:21	20:16	15:11	10:6	5:0

a. R-type instruction

Field	35 or 43	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

b. Load or store instruction

Field	4	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

c. Branch instruction

OPCODE:

Opcode is a field that denotes the operation and format of an instruction.

R-TYPE INSTRUCTION

R-format instruction has opcode value as 0. It has three register **rs, rt** and **rd**. Fields **rs** and **rt** are used for sources and rd is for destination.

ALU function is in the function field.

R-type instruction will support **add, sub, AMD, OR** and **slt** operations, **shamt** field is used only for shifts.

LOAD OR STORE INSTRUCTIONS

Load or store instruction has opcode value as 35 or 43.

- Load – 35
- Store – 43

Register **rs** is the base register that is added to the **16 bit address** field to form the memory address.

For loads, **rt** is the **destination register** for the loaded value.

For store **rt** is the **source register** whose value should be stored into memory.

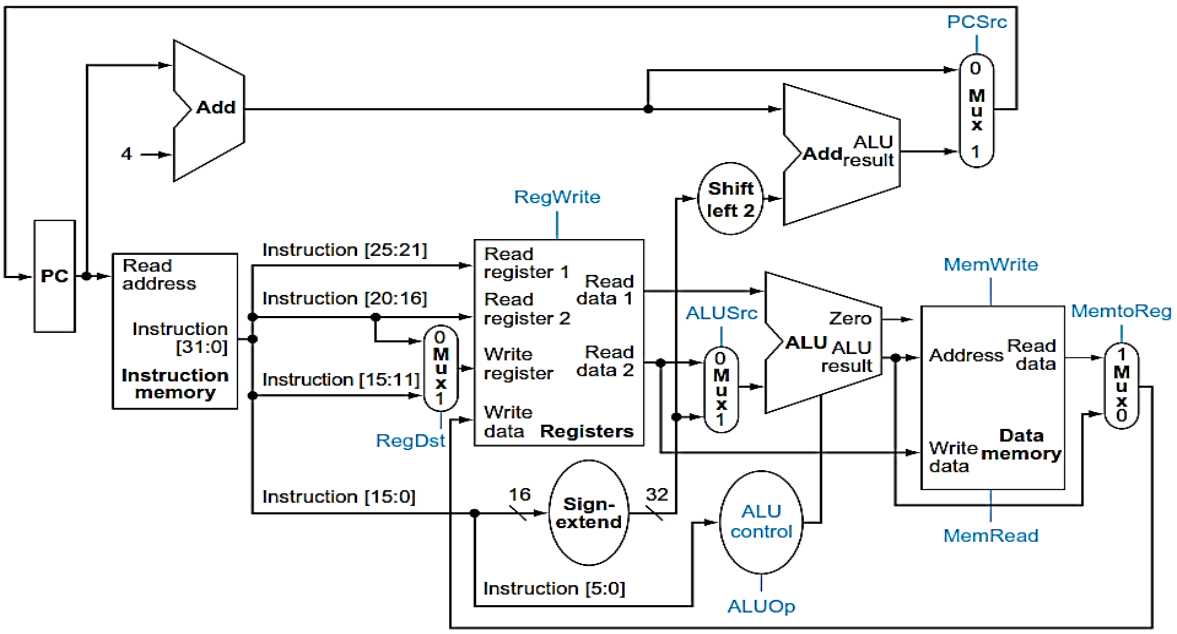
BRANCH INSTRUCTION

Branch instruction has opcode value as 4. Register rs and rt are the source registers that are compared for equality. The 16-bit address field is sign extended, shifted and added to the PC + 4 to compute the branch target address.

These are several major observations about this instruction format such as

[1] The Opcode field is always contained in bits **31:26**.
[2] The two registers to be read are always specified by the rs and rt fields at positions **25:21** and **20:16**. This is true for R-type instructions, branch equal and store.
[3] The base register for load and store instruction is always in bit positions **25:21** (rs).
[4] The 16-bit offset for branch equal, load and store is always in positions **15:0**.
[5] The destination register has two places.
  a. For load it is in bit position **20:16** (**rt**)
  b. For R-type it is in bit positions **15:11** (**rd**)

So we need to add a multiplexer to select which field of the instruction is used to indicate the register number to be written. Using this information, we can add the instruction labels and extra multiplexer to the simple data path.



The above figure shows the ALU control block

- ❑ Write signals for state elements
- ❑ Read signal for data memory

- Control signals for the multiplexers.

All the multiplexers have two inputs so each require a single control line.

PC does not require a write control, since it is written once at the end of every clock cycle.

The branch control logic determines whether it is written with the incremented PC or the branch target address.

It shows seven single bit control line and 2-bit ALUOp control signal.

5.4. THE EFFECT OF THE SEVEN CONTROL SIGNALS

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the write register comes from the rt field (bits 20:16)	The register destination number for the write register comes from the rd field, bits (15:11),
Regwrite	None	The register on the write register input is written with the value on the write data input.
ALUsrc	The second ALU operand ewes from the second register file output (read data 2)	The second ALU operand is the sign extended, lower 16 bits of the instruction.
PCsrc	The PC is replaced by the output of the adder that computes tfee value of PC+ 4	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None	Data memory contents designated by address input are put on the read data output.
Memwrite	None	Data memory contents designated by the address input are replaced by the value on the write data input.
MemtoReg	The value fed to the register write data input comes from the ALU	The value fed to the register write data input comes from the data memory.

6. PIPELINING

Pipelining is an implementation technique in which multiple instructions are overlapped in execution. Pipelining is the most important technique used to increase the speed mid performance of the processor.

To execute a MIPS instruction through the pipeline it has five steps.

[1] Fetch instruction from memory
[2] Read registers while decoding the instruction. Tie regular format of MIPS instructions allows reading and decoding to occur simultaneously.
[3] Execute the operation or calculate an address
[4] Access an operand in data memory.

[5] Write the result into a register.

6.1. SINGLE CYCLE VS PIPELINED PERFORMANCE

Consider three kinds of different instruction classes are

- Load and store word instruction
- R-type instructions
- Branch instructions

From these three instruction classes we have eight instructions such as load word (Iw), store word (sw), add (add), subtract (sub), AND, OR, set less than (slt) and branch on equal (beq).

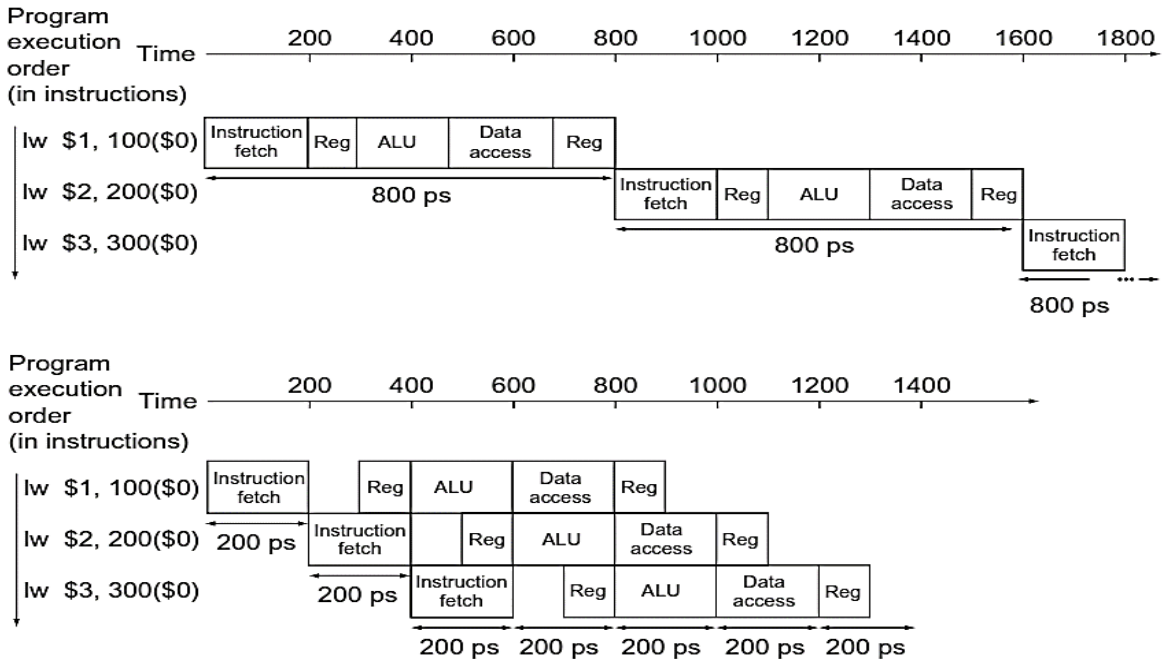
For these eight instructions we have to find the average time taken to execute the instructions in both single cycle and pipelined method.

Assume operation times for major functional units.

- For memory access 200 ps
- For ALU operation 200 ps
- For register file read or write 100 ps

In single clock cycle model every instruction takes exactly one clock cycles. So it will produce the slow speed to execute the instruction.

The figure below shows how single cycle instructions are executed in non-pipelined- execution and pipelined execution for three load word instructions.



Both use the same hardware components, whose time is listed below,

The time between the first and fourth instructions in the non-pipelined design is 3x800=2400 ps.

The time between the first and fourth instructions in the pipelined design is 3x200=600 ps.

$$Time\ \ between\ instructions_{pipelined} = \frac{Time\ \ between\ instructions_{non-pipelined}}{No.\ of\ Pipeline\ Stages}$$

Pipelining improves performance by increasing instruction throughput and decreasing the execution time of an individual instruction.

6.2. Designing Instruction Sets For Pipelining

In pipelining process instruction sets has the more importance because real programs execute billions of instructions so we need to increase the throughput of instruction.

For designing instruction sets for pipelining process we have to consider some important factor such as

- Length of the instruction
- Instruction format
- Memory operands
- Operand alignment

To explain the pipelining process we take the MIPS instruction set, so using these four factors we can design for pipelined execution.

6.2.1. Length of The Instruction

All MIPS instructions are same length. It makes much easy to fetch instructions in the first pipeline stage and to decode them in the second stage.

6.2.2. Instruction Format

MIPS has only a few instruction formats. So source register field is located in the same place in all instructions.

If instruction format is symmetry then the second stage can begin reading the file at the same time the hardware is determining what type of instruction was fetched.

If instruction format is not symmetry then we have to split the second stage into two parts. It will increase the stages of pipelining process.

6.2.3. Memory Operands

MIPS instruction set has only two memory operand **(lw and sw).**

It can use the execute stage to calculate the memory address and then access memory in the following stage.

6.2.4. Operand Alignment

Operands must be aligned in memory. So, we need not worry about a single data transfer instruction requiring two data memory accesses.

The requested data can be transferred between processor and memory in a single pipeline stage.

7. Pipeline Hazards

There are situations in pipelining when the next instruction cannot execute in the following clock cycle. These events are called hazards.

Types of hazards

There are three different types of hazards

- Structural hazards
- Data hazards
- Control hazards

7.1. Structural Hazards

It means when a planned instruction cannot execute in the proper clock cycle because the hardware does not support the combination of instructions that are set to execute.

In MIPS instruction set structural hazard will appear in following case:

In the same clock cycle the first instruction is accessing data from memory and the fourth instruction is fetching an instruction from, that same memory. So it cause structural hazard.

7.2. Data Hazards

Data hazard occur when a planned instruction cannot execute in the proper clock cycle because data that is needed to execute the instruction is not yet available.

In pipeline process if one step must wait for another to complete means it cause data hazards. Data hazard mainly occur if one instruction depends on another instruction to complete their task.

7.2.1. Example

Suppose we have an add instruction followed immediately by a subtract instruction that uses the sum ($s0).

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

Subtract instruction has to wait until the add instruction is executed. Because it has to get $s0 value from the add instruction itself.

Without intervention, a data hazard could severely stall the pipeline.

The add instruction does not write its result until fifth stage, it means we have to waste three clock cycle in the pipeline.

In fifth clock cycle, only add write the result but subtract has to read the value in second clock cycle, itself.

7.2.2. SOLUTION FOR DATA HAZARD

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.

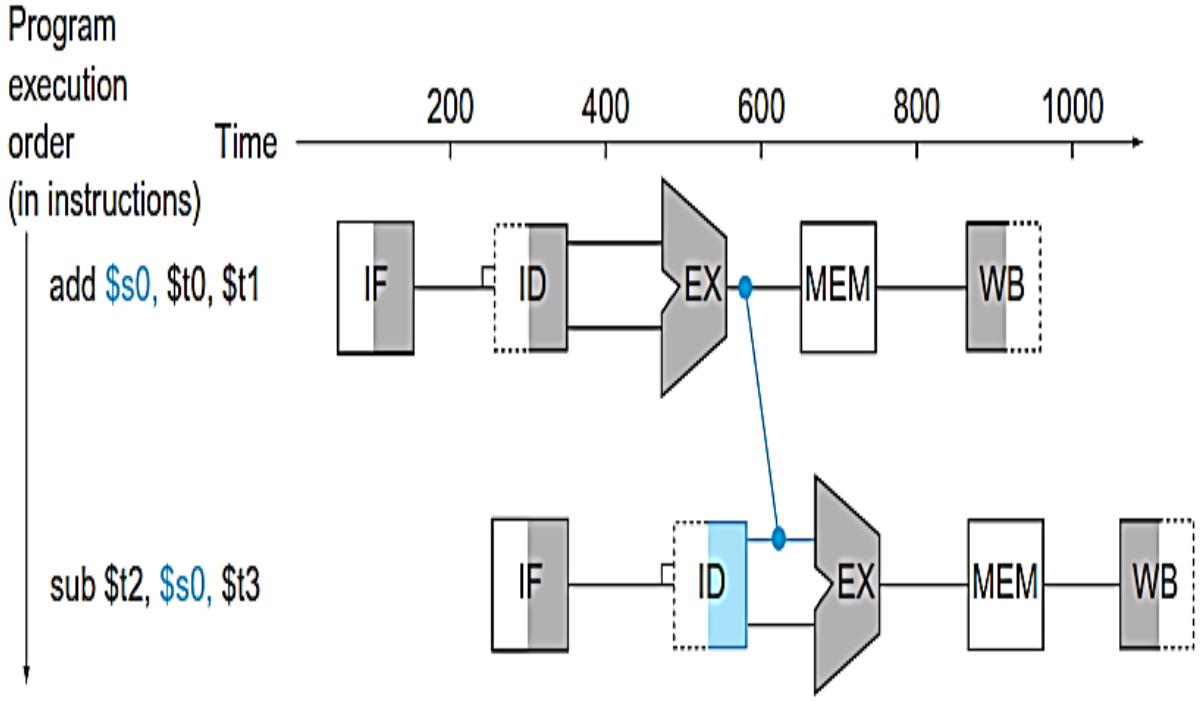
For that above example as soon as the ALU executes the sum for the add instruction, we can supply it as an input for the subtract instruction. Adding extra hardware to retrieve the missing item early from the internal resources is called ***forwarding*** or ***bypassing***.

7.2.2.1. FORWARDING

It is also called as bypassing. It is a method of resolving a data hazard by retrieving the missing data element from internal buffers rather than waiting for it to arrive from programmer visible registers or memory;

EXAMPLE FOR FORWARDING

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



Generally in MIPS instruction set has five stages for pipelining process.

- IF - Instruction field
- ID - Instruction decode/register file read stage
- EX - Execution stage
- MEM - Memory access stage
- WB - Write back stage

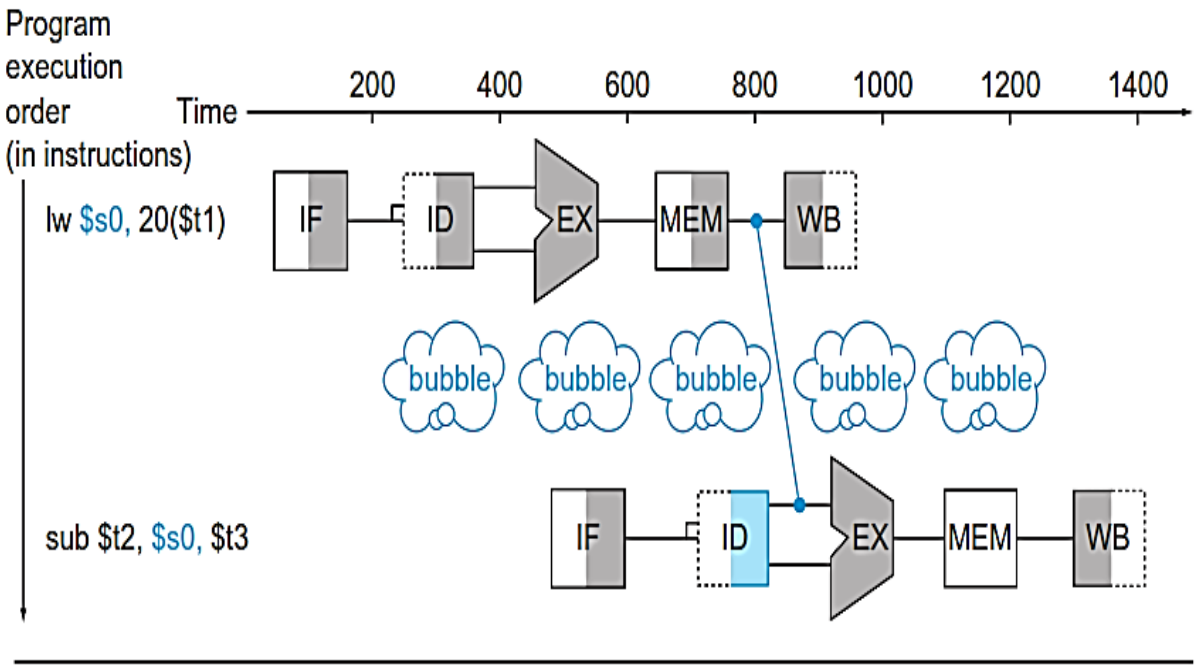
7.2.3. LOAD USE DATA HAZARD

It is a specific form of data hazard in which the data being loaded by a load instruction has not yet become available when it is needed by another instruction.

7.2.4. PIPELINE STALL

It is also called bubble.

Pipeline stall is a stall initiated in Order to resolve a hazard. It is shown in figure below.



7.3. CONTROL HAZARDS

The third type of hazard is called control hazard, arising from the need to make a decision based on the results-of one instruction while others are executing.

Control hazard is also called branch hazard. When the proper instruction cannot executing the proper pipeline clock cycle because the instruction was fetched is not the one that is needed.

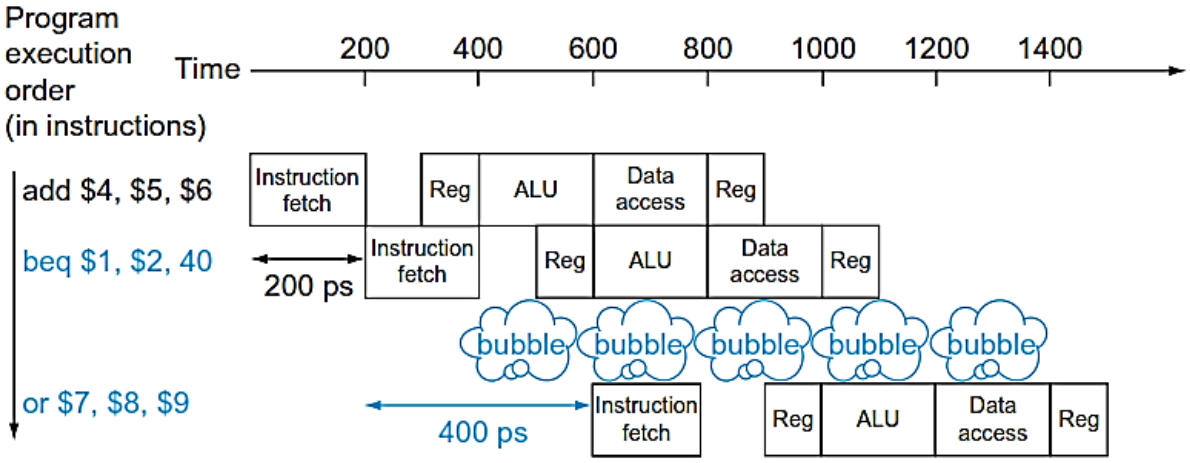
That is the flow of instruction addresses is not what the pipeline expected.

Consider the branch instruction, we must begin fetching the instruction following the branch on the very next clock cycle.

The pipeline cannot possibly know what the next instruction should be, because it only received the branch instruction from memory.

To avoid stall, we fetch a branch after that waiting until the pipeline determines the outcome of the branch and knows what instruction address to fetch from. Let's assume that we put in enough extra hardware so that we can test registers, calculate the branch address and update the PC during the second stage of the pipeline.

Even with the extra hardware, the pipeline involving conditional branches as like below figure,



7.3.1. BRANCH PREDICTION

Branch prediction is a method of resolving a branch hazard that assume a given outcome for the branch and proceeds from that assumption rather than waiting to ascertain the actual outcome.

7.3.1.1. DYNAMIC HARDWARE PREDICTORS

Dynamic hardware predictors make their guesses depending on the behaviour of each branch and may change predictions for a branch over the life of a program.

One popular approach to dynamic prediction of branches is keeping a history for each branch as taken or untaken.

Then using the recent past behaviour to predict the future behaviour.

7.4. ADVANTAGES OF PIPELINE

It increases the number of simultaneously executing instructions. It increases the rate at which instructions are started and completed. It improves instruction throughput rather than individual instruction execution or latency.

7.5. LATENCY

Latency is the number of stages in a pipeline or the number of stages between two instructions during execution.

8. PIPELINE DATAPATH AND CONTROL

Pipelining has five stages.

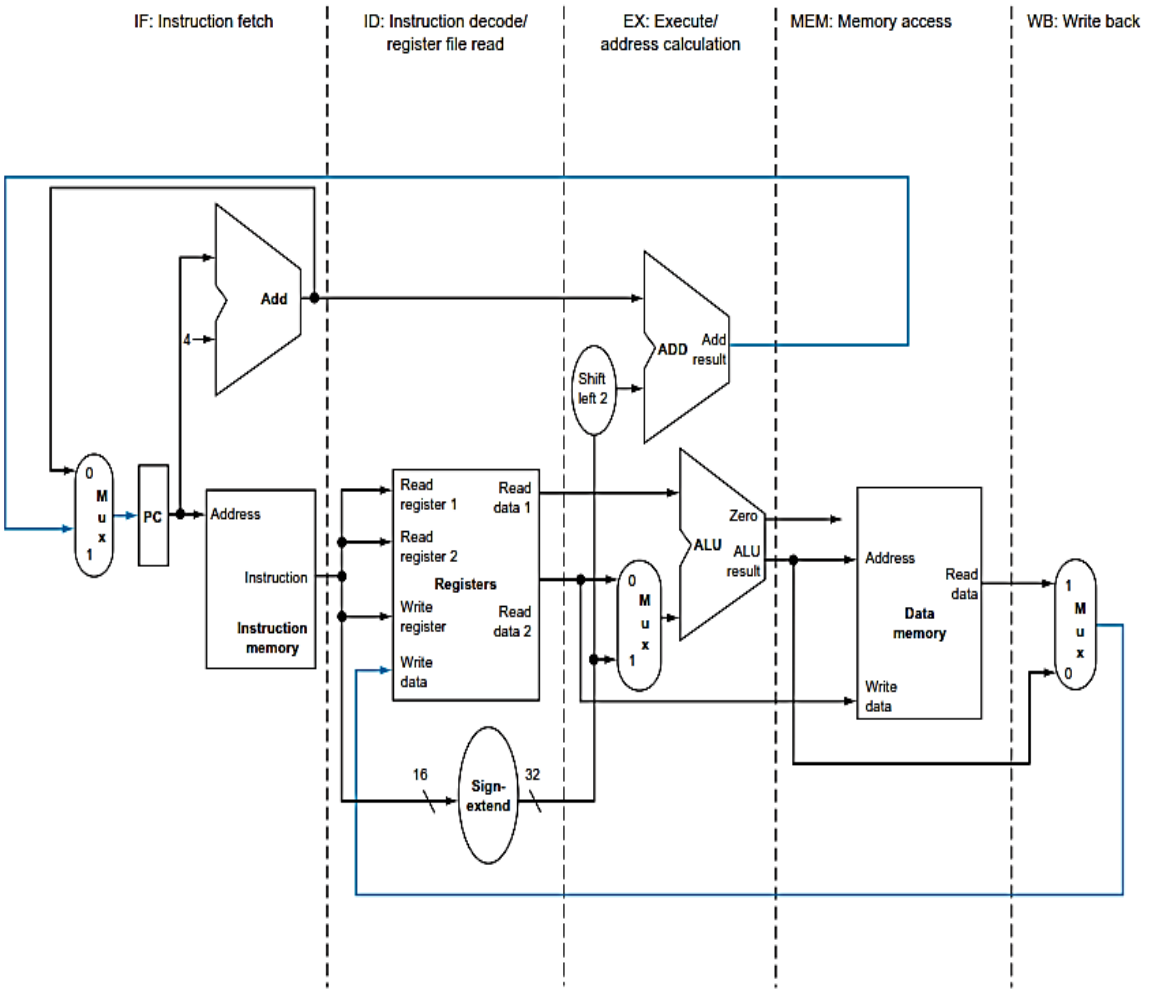
- IF: Instruction fetch
- ID: Instruction decode and register file read
- EX: Execution or address calculation
- MEM: Data memory access
- WB: Write back

Instruction and data move are generally from left to right through the five stages to complete execution.

There are some Cases we cannot perform this' left to right movement that exceptions cases are

[1] The write back, which places the result back into the register file in the "middle of the datapath.
[2] The selection of the next value of the PC, choosing between the incremented PC and the branch address from the MEM stage.

These two process stage will perform data flowing from right to left. Data flowing from right to left does not affect the current instruction. It will reverse data movements influence only later instructions in the pipeline. The below figure shows the pipelining process for single cycle datapath.



First right to left flow of data can lead to data hazards. Second right to left flow of data can lead to control hazards. Consider three load word instructions and how datapath are created during pipelining process.

Three load word instructions are

- lw $l, 100($0)
- lw $2,200 ($0)
- lw $3,300 ($0)

Each stage is labelled by the physical resource used in that stage. IM represents the instruction memory and the PC in the instruction fetch stage. To maintain proper time order, this datapath break the register file into tw0 logical parts

[1] Registers read during register fetch (ID)
[2] Registers written during Write back (WB)

The register file is written in the first half of the clock cycle and the register file is read during the second half.

8.1. PIPELINED DATAPATH

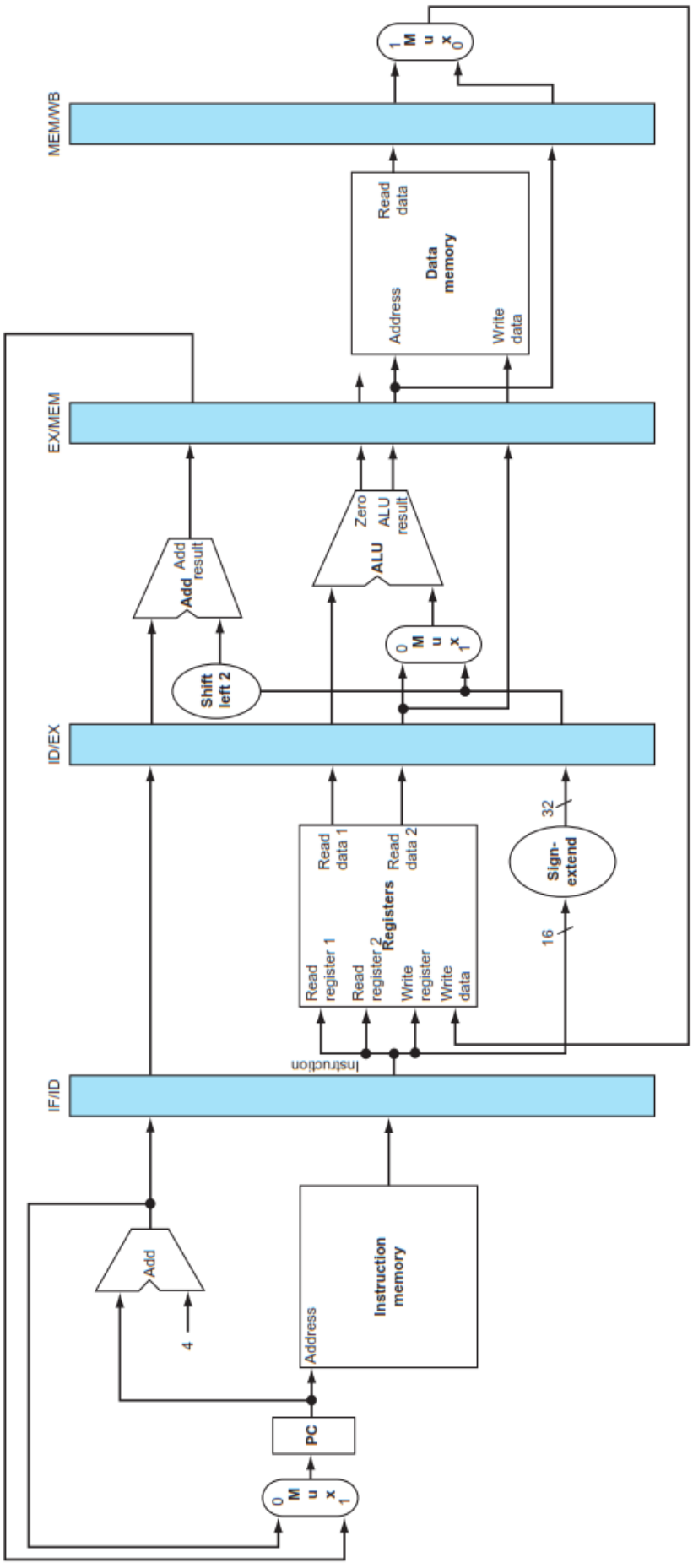
Pipelined datapath can be implemented using the pipeline registers. All instructions advance during each clock cycle from one pipeline register to the next. The registers are named for the two stages separated by that register.

For example the pipeline registers between IF and ID stages is called IF/ID.

There is no pipeline register at the end of the write back stage. All instruction must update some state in the processor.

The register file, memory or the PC has a separate pipeline register is redundant to the state that is updated. Every instruction updates the PC whether by incrementing it or by setting it to a branch destination address.

PC is part of the visible architecture state and its contents must be saved when an exception occurs, while the contents of the pipeline registers can be discarded.


MEM/WB
EX/MEM
ID/EX
IF/ID
M u x
Read data
Data memory
Address
Write data
Zero
ALU
ALU result
Add
Add result
Shift left 2
Registers
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Sign-extend
32
16
Instruction
Instruction memory
PC
4

8.2. PIPELINED CONTROL

Pipeline can be implemented in two ways:

[1] Single cycle implementation
[2] Multiple cycle implementation

To provide control in the pipelined datapath the following task has to be followed.

STEP 1: Label the control lines on the existing datapath.

STEP 2: Borrow the Control from the simple datapath. ,

STEP 3: Use same ALU control logic, branch logic, destination register number multiplexer and control lines used in simple datapatH.

8.3. PIPELINED CONTROL FOR SINGLE CYCLE IMPLEMENTATION

In case of single cycle implementation assume that the PC is written on each clock cycle. So no need to provide separate write signal for the PC.

If we are implementing pipeline, control in single cycle implementation for that also no need separate write signals.

Because pipeline registers also written during each clock cycle. To specify control for the pipeline we have to set the control values during each pipeline stage.

Because each control line is associated with a component active in only a single pipeline stage. We have five stages for pipeline such as

[1] Instruction fetch
[2] Instruction decode/register file read
[3] Execution/address calculation
[4] Memory access
[5] Write back

Overall pipeline control has divided into five groups according to the pipeline stage.

8.3.1. INSTRUCTION FETCH

The control signals to read instruction memory and to write the PC are always asserted.

So there is nothing special to control in this pipeline stage. For instruction fetch stage we don't want special control line.

8.3.2. INSTRUCTION DECODE/REGISTER FILE READ

As like instruction fetch stage, here also same thing happens at every clock cycle.

So in this stage also we no need to set any control lines

8.3.3. EXECUTION/ADDRESS CALCULATION

In this stage we have to use some control lines for example RegDst, ALUOp and ALUsrc. These control lines are used to select the result register, ALU operation and either read data or sign extended immediate for the ALU.

8.3.4. MEMORY ACCESS

In this stage the following control lines are used

[1] Branch
[2] MemRead
[3] Memwrite

The branch equal, load and store instructions are used above control lines.

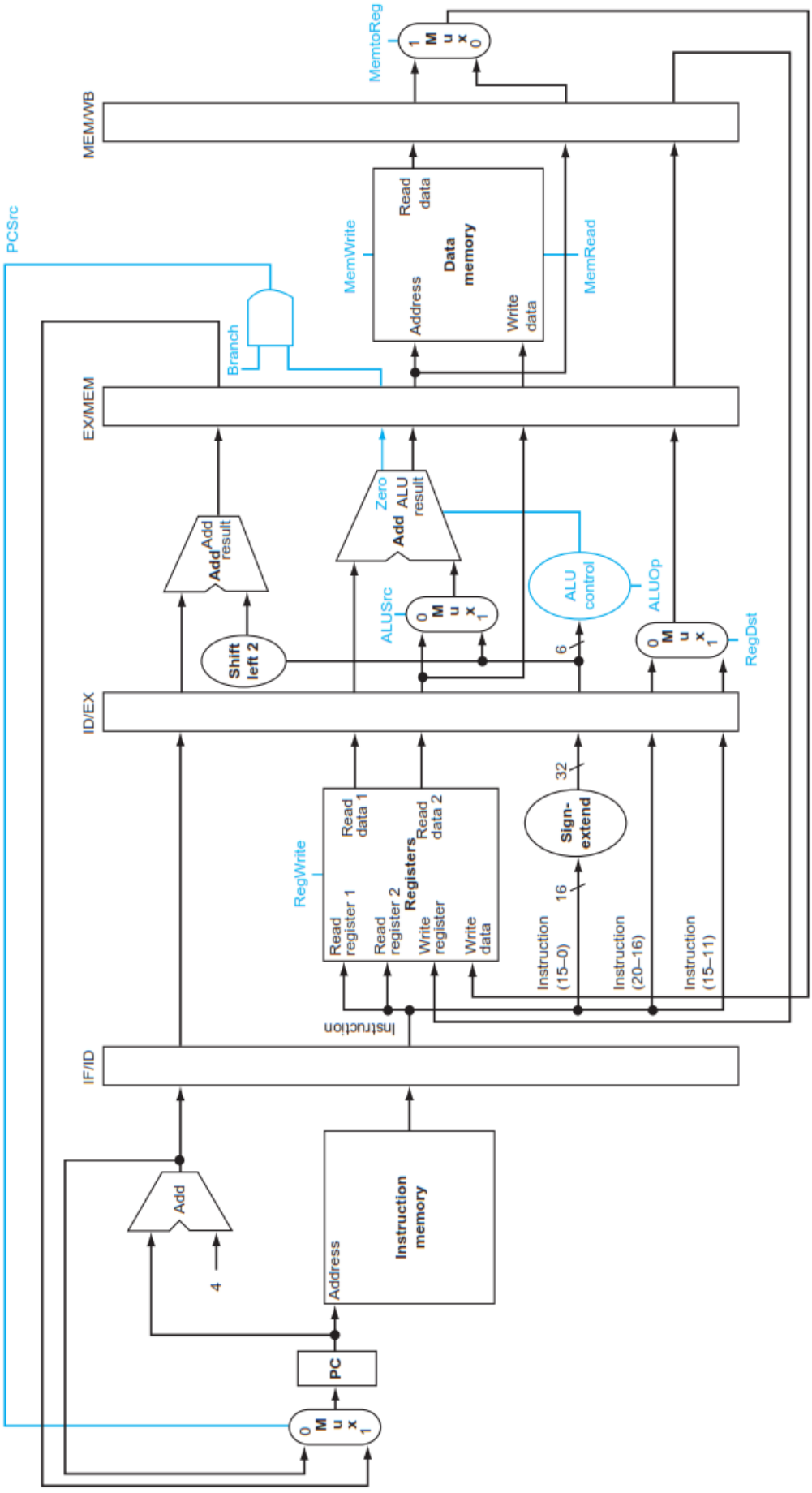
8.3.5. WRITE BACK,

In this stage two control lines are used such as MemtoReg and Regwrite.

MemtoReg control line decides between the ALU result or the memory value to the register file and regwrite writes the chosen value. .

For pipeline control we have seven control lines that are used in five stages of pipeline process. Implementing control line is the process of setting the nine control lines value in each stage for each instruction. Control lines start with the EX stage so we have to create the control information during instruction decode stage.

The below figure shows how these seven control lines are implemented in the pipelined datapath.


MemtoReg
M u x
MEM/WB
Read data
Data memory
Address
Write data
MemWrite
MemRead
PCSrc
Branch
EX/MEM
Zero
ALU
Add
Add result
ALU result
ALUSrc
ALU control
ALUOp
RegDst
Shift left 2
6
ID/EX
32
Sign-extend
RegWrite
Read data 1
Read data 2
Registers
Read register 1
Read register 2
Write register
Write data
16
Instruction (15–0)
Instruction (20–16)
Instruction (15–11)
Instruction
IF/ID
Add
4
Address
Instruction memory
PC

9. DATA HAZARDS AND STALLS

Forwarding method used to resolve the problem occurred in data hazards. Consider the following set of instructions.

```
lw$2,20 ($1)
and $4, $2, $5
or $8, $2, $6
add $9, $4, $2
slt $1, $6, $7
```

Here dependence between the load and following instruction goes backward in time, this hazard cannot be solved by forwarding technique. So in addition to a forwarding unit, we need one more new unit called hazard detection Unit.

Hazard detection unit operated during the ID stage so it can insert the stall between the load and its use.

Hazard detection unit has single condition to provide control to load instructions.

```
if (ID/EX.MemRead and
        ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
        (ID.EX.REgisterRt = IF/ID.RegisterRt)))
                stall the pipeline
```

First line test if the instruction is a load or not. Load instructions only can read data memory.

The next two lines check if destination register field of the load in the EX stage matches either source register of the instruction in the ID stage.

If these conditions hold, the instruction stalls one clock cycle.

After this 1 cycle stall, the forwarding logic can handle the dependence and execution proceeds.

- If there is no forwarding logic then the above instruction need another stall cycle.
- If the instruction in the ID stage is stalled, then the instruction in the IF stage must also be stalled.
- If IF stage not stalled means we lose the fetched instruction.

To avoid .stalled in these two stage we can prevent the PC registers and the IF/ID pipeline register from changing.

The instruction in the IF stage will continue to read using the same PC, The registers in the ID stage will continue to read using the same instruction fields in the IF/ID pipeline register. .

9.1. NOP [NO OPERATION]

An instruction that does no operation to change state. NOP is a case where pipeline stage executing an instruction but that does not make any changes in the state.

NOP acts like a bubble in the pipeline stage.

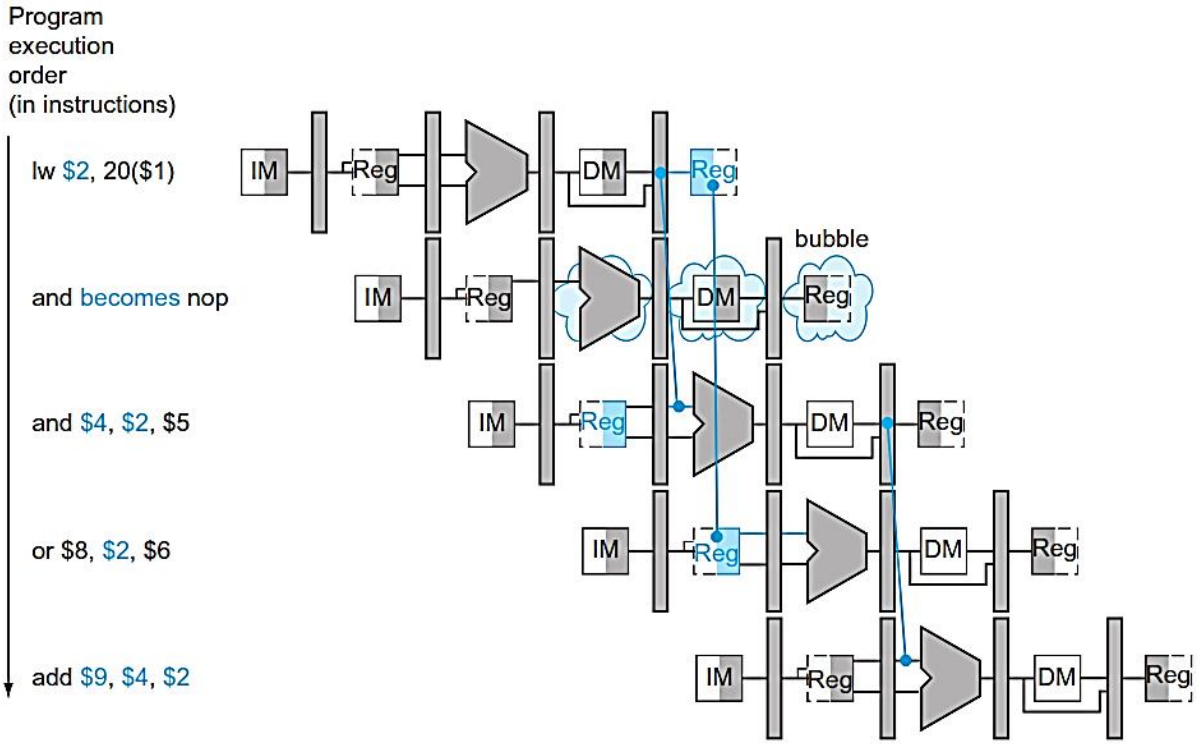
9.2. INSERTING NOPS IN PIPELINE.

NOPS is like bubble, to identify hazard in ID stage we had to insert bubble into the pipeline.

Once the bubble is inserted in the ID stage it will change the control field of EX, MEM and WB field of the ID/EX pipeline register to 0. If the control values are 0 then no registers or memories are written. It knows the pipeline execution slot associated with the AND instruction is turned into a nop.

If AND instruction is nop then all instructions beginning with the AND instruction are delayed one cycle.

A bubble is inserted beginning in clock cycle 4, by changing the AND instruction to a nop.



AND instruction is really fetched and decoded in clock cycle 2 and 3 but its EX stage is delayed until clock cycle 5.

OR instruction is fetched in clock cycle 3 but its ID stage is delayed until clock cycle 5.

After inserting bubble, all the dependencies go forward in time and no further hazards occur.

9.3. HAZARD DETECTION UNIT

This unit controls the writing of PC and IF/ID registers.

It also controls the multiplexer that chooses between the real control values and all O's.

9.4. FORWARDING UNIT

This unit controls the ALU multiplexers to replace the value from a general purpose register with the value from the proper pipeline register.

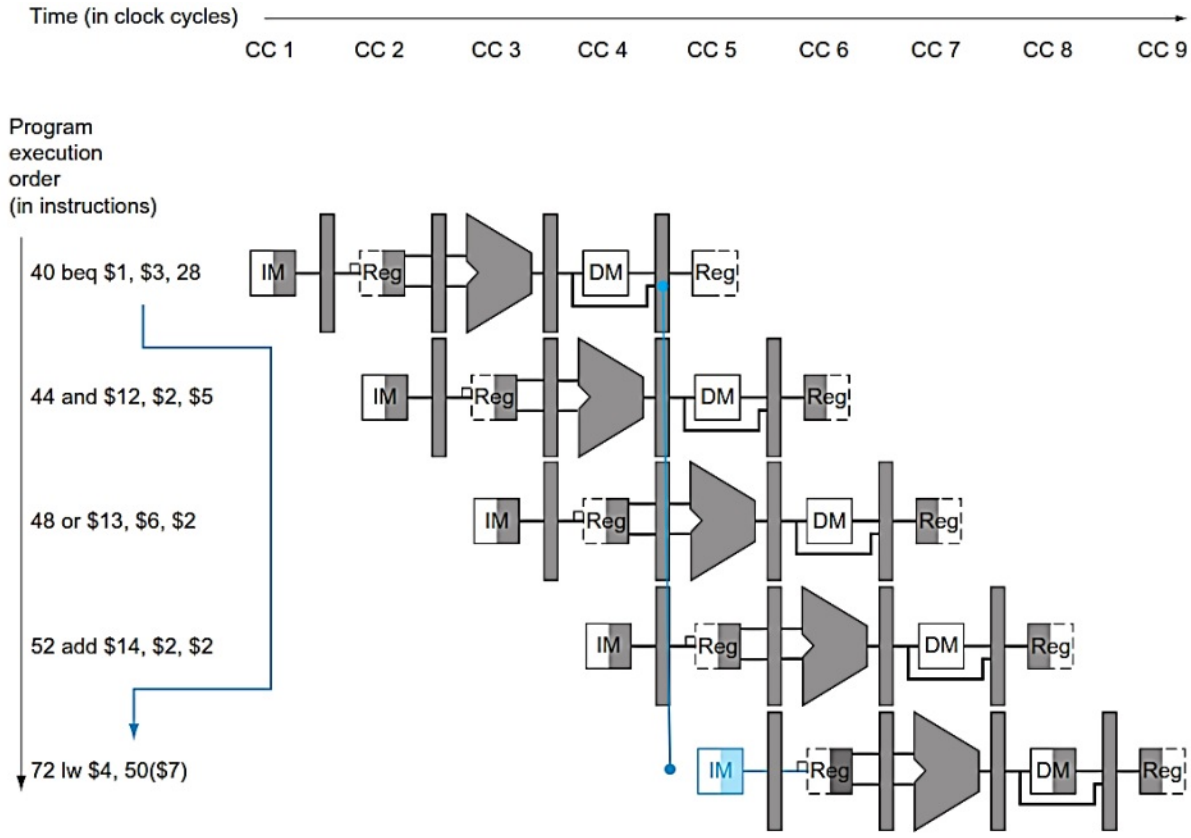
Using these two unit we can resolve the data hazards occurred in pipelining

10. CONTROL HAZARD

Control hazard occurs when we execute the branch instruction in pipeline process. Control hazard are relatively simple to understand and it occurs at less frequently. To avoid control hazard in pipeline process we can use simple method and no need to go for any special techniques like data hazard.

Let us consider the following sequence of instructions and how control hazards occur in that.

The figure below shows a sequence of instructions, and indicates when the branch occurs in this pipeline.



An instruction must be fetched at every clock cycle to sustain the pipeline. The decision about whether to branch does not occur until the MEM pipeline stage.

The numbers to the left of the instruction (40, 44, 48, 52, 72) indicates the address of the instruction.

beq instruction is executed at MEM stage only but before that and, or, add instructions are executed

To resolve the control hazard we have to know whether branch not taken or not. There are two schemes used to resolve the control hazards such as

- Branch not taken
- Branch prediction

10.1. BRANCH NOT TAKEN

If the branch is taken then the instructions that are being fetched and decoded must be discarded. After discarding the execution continues at the branch target.

Discarding instructions means we must be able to flush instructions in the IF, ID and EX stages of the pipeline. Flush is a method used to discard instructions in a pipeline usually due to an unexpected event

If branch instruction is fetched immediately it stall the execution until the pipeline determines the outcome of the branch and knows what instruction address to fetch from.

If branch not taken means no need to discard any instructions and pipelining will execute the instructions continuously.

Branches are taken then only pipeline has stalled so by reducing the delay of branches we can improve the performance of pipelining process in this condition.

10.2. BRANCH PREDICTION

Branch prediction is a method of resolving a branch hazard. It assumes a given outcome for the branch and proceeds from that assumption rather than waiting to ascertain the actual outcome.

A simple form of branch prediction is we have to assume branch is not taken.

This assumption is possible only simple five, stage pipeline.

For deeper pipelines this assumption is not suitable because it will increase the branch penalty when measured in clock cycles.

For such kind of pipelines we have to add more hardwire to predict branch behaviour during program execution.

Solution for increasing branch penalty we have new technique called ***dynamic branch prediction***.

10.2.1. DYNAMIC BRANCH PREDICTION

Dynamic branch prediction is a prediction of branches using runtime information.

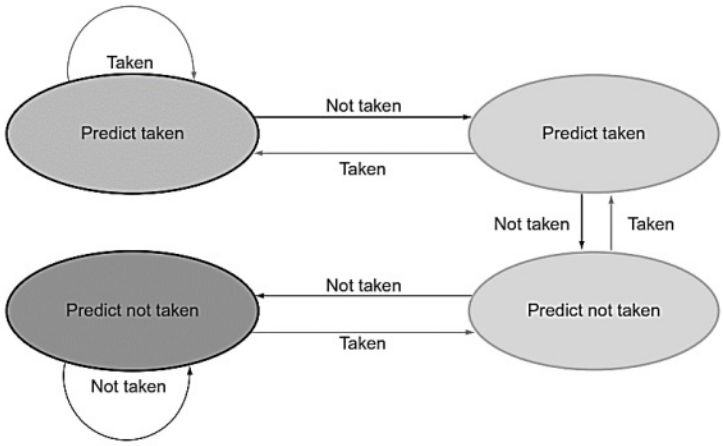
To implement dynamic branch prediction method we have to use one buffer that is called branch prediction buffer.

Branch prediction buffer also called branch history table.

This buffer is small memory that is indexed by the lower portion of the address of the branch instruction.

It also contains one or more bits indicating whether the branch was recently taken or not. Prediction is a method that assumes correct action, so fetching in the predicted direction. If the prediction assumption is wrong then the incorrectly predicted instructions are deleted.

The prediction bit is inverted and stored back then the proper sequence is fetched and executed.



11. EXCEPTIONS IN MIPS

Control is the most challenging aspect of processor design because Of two reasons:

[1] It is the hardest part to get right.
[2] It is the hardest part of make fast.

One of the hardest parts of control is implementing exceptions and interrupts. Handling exceptions and interrupts are more complex task than handling branches or jumps.

Exception and interrupt are initially created to handle unexpected events from Within the processor

11.1. EXCEPTION

Exception is an unscheduled event that disrupts program execution and used to detect overflow. It is also called interrupt.

11.2. INTERRUPT

Interrupt is an exception that comes from outside of the processor. We must distinguish between interrupts mid exception is more important. Exception refers to any unexpected change, in control flow without distinguishing whether the cause is internal or external. Interrupt refer to any unexpected^ change occurred only when the event is externally caused.

Type of event	From where?	MIPS terminology
I/O device request	External	Interrupt
Invoke the operating system from user program	Internal	Exception
Arithmetic overflow	Internal	Exception

Using an undefined instruction	Internal	Exception
Hardware malfunctions	Either	Exception or interrupt

11.3. EXCEPTIONS IN MIPS ARCHITECTURE

In MIPS architecture two kinds of exceptions occur at

[1] Execution of an undefined instruction
[2] An arithmetic overflow

Once exception occur then fee processor must save fee address of the offending instruction in the exception program counter (EPC) and then transfer control to the operating system at some specified address.

After transferring control to fee operating system it must take appropriate action to provide some service to the user program.

Operating system will provide the following services to user program.

[1] Taking some predefined action in response to an Overflow.
[2] Stopping the execution of the program and reporting an error.

After performing necessary action is required because of exception, fee operating system can terminate fee program or may continue its instruction. Operating system use the exception program counter (EPC) to determine where to restart fee execution of the program.

To handle the exceptions by operating system it must know the reason for the exception.

Operating system must know the reason for exception because then only it can handle properly.

There are two main methods used to communicate the reason for an exception. MIPS architecture has following two methods to find the reason for exception

[1] Status register
[2] Vectored interrupts

11.3.1. STATUS REGISTER

It is also called the cause register. It holds a field that indicates the reason for the exception.

11.3.2. VECTORED INTERRUPTS

In a vectored interrupt, the address to which control is transferred is determined by the cause of the exception.

11.4. IMPLEMENTATION OF EXCEPTION IN MIPS ARCHITECTURE

To implement exception system in MIPS architecture assume it has single entry point for all exception and has address value is 8000 0180. This address value indicates it is an arithmetic overflow exception.

To handle this exception we need to add two additional registers to our current MIPS implementation.

Two additional registers are one is EPC and another is cause register.

11.4.1. EPC

A 32 bit register used to hold the address of the affected instruction. This register is needed even when exceptions are vectored.

11.4.2. CAUSE REGISTER

This register is used to record the cause of the exception.

In MIPS architecture this register is 32 bits long and some bits are currently unused.

Assume that there is a five bit field that encodes the undefined instruction exception and arithmetic overflow exception.

For these two kinds of exceptions 10 bits are used to represent an undefined instruction and 12 bits used to represent arithmetic overflow.

PART – A

BASIC MIPS IMPLEMENTATION

1. **What are the instructions set available in MIPS architecture?**

   There are three kinds of instruction sets available in MIPS architecture.

   1. Memory reference instruction set Ex-load word or store word
   2. Arithmetic logical instruction set Ex-add, sub, AND, OR and slt
   3. Branch and jump instruction Ex-branch equal and jump

2. **What is meant by program counter?**

   Program counter is a register containing the address of the instruction in the program being executed.

3. **Define register file.**

   Register file is a structure where processor's 32 general purpose registers are stored.

   Register file is a collection of registers that can be read and written by specifying the number of register in the file.

4. **What are the units needed to implement load and store instructions?**

   Four units are needed to implement MIPS load and store instructions such as

   - Register file
   - ALU
   - Data memory unit
   - Sign extension unit

5. **Define truth table.**

   It is a logical representation for operation by listing all the values of the inputs and showing what the resulting outputs should be.

6. **What is meant by load use data hazard?**

   It is a specific form of data hazard in which the date being loaded by a load instruction has not yet become available when it is needed by another instruction.

7. **What is meant by cause register?**

   Status register also called as cause register.

   It holds a field that indicates the reason for the exception.

BUILDING DATAPATH

8. **What is meant by datapath element?**

   Data path element is a unit used to operate on or hold data within a processor.

In the MIPS implementation, the data path elements include the instruction and data memories, the register file, ALU and adders.

9. What is data path?[Nov/Dec-2016]

A data path is a collection of functional units, such as arithmetic logic units or multipliers that perform data processing operations, registers and buses. Along with the control unit it composes the central processing of CPU.

10. What is meant by sign extend unit?

Sign extend unit increases the size of data item by replicating the high order sign bit of the original data item in the high order bits of the larger, destination data item.

PIPELINING

11. What is meant by pipelining?

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

12. Mention the various types of pipelining. [Nov/Dec-2017]

- Arithmetic pipelining.
- Instruction pipelining.

13. What is meant by pipeline bubble?[Nov/Dec-2016]

In computing bubble or pipeline stall is delay in execution of an instruction in an instruction pipeline in order to resolve a hazard.

14. What are the advantages of pipelining?[April/May-2016]

The advantages of pipelining is that the cycle time of the processor is reduced and it can increase the instruction throughput.

15. How many stages available in pipelining datapath?[Nov/Dec-2017]

There are five stages available in pipelining datapath.

- **IF:** Instruction Fetch V.
- **ID:** Instruction Decode and Register File Read
- **EX:** Execution or address calculation
- **MEM:** Data Memory Access -
- **WB:** Write Back

16. What are the ways that pipelining can be implemented?

Pipelining can be implemented in two ways:

1. Single cycle implementation
2. Multiple cycle implementation

17. What is meant by hazard in pipelining?

In pipelining there are situations when the next instruction cannot execute in the following clock cycle. These events are called hazard.

18. How many types of hazards can occur in pipelining?

There are three hazards present in the pipelining such as

1. Structural hazard
2. Data hazard
3. Control hazard

19. What is meant by dynamic branch prediction?

Dynamic branch prediction is a prediction of branches at runtime using runtime information.

20. What is meant by branch prediction buffer? *[April/May 2015]*

Branch prediction buffer is also called as branch history table.

It is a small memory that is indexed by the lower portion of the address of the branch instruction.

It contains one or more bits indicating whether the branch was recently taken or not.

21. What is meant by branch target address?

It is an address specified in a branch which becomes the new program counter (PC) if the branch is taken.

22. Distinguish pipelining from parallelism. *[April/May 2015]*

Parallelism is the simultaneous execution of more than one task with the help of multiple processors working in coordination with one another.

Pipelining is the process of increasing the throughput by utilizing various parts of the processor in coordination with one another without wasting clock cycles.

23. What is meant by latency in pipeline?

Latency is a number of stages in a pipeline or the number of stages between two instructions during execution.

24. What are R-type instructions?[April/May-2015]

- Add
- Sub
- AND
- OR
- Slt.

HANDLING DATA & CONTROL HAZARDS

25. Define hazard. Give an example for data hazard.[April/May-2017]

In Pipelining there are situations when the next instruction cannot execute in the following clock cycle. The events are called hazard.

26. Define structural hazard.

Structural hazard means the hardware cannot support the combination of instructions that we want to execute in the same clock cycle.

27. Define data hazard.

Data hazard occur when a planned instruction cannot execute in the proper clock cycle because data that is needed to execute the instruction is not yet available

Data hazard mainly occur in executing arithmetic instructions.

28. Define control hazard.

Control hazard also called branch hazard, It occurs when the flow of instruction addresses is not what the pipeline expected.

29. How to resolve a hazard?

Pipeline stall also called bubble. A stall can be initiated in order to resolve a hazard. .

30. How data hazards are resolved?

Forwarding method is used to resolve the data hazards. It is also called bypassing.

Forwarding is a method of resolving data hazard by retrieving the missing data element from internal buffers rather than waiting for it to arrive from programmer visible registers or memory.

31. How control hazards are resolved?

Control hazard or branch hazard pan be resolved using branch prediction method.

Branch prediction is a method of resolving a branch hazard that assumes a given outcome for the branch and proceeds from that assumption rather than waiting to ascertain the actual outcome.

EXCEPTIONS

32. What is meant by exception? [Nov/Dec 2014] [April/May-2018] ***[April/May-2016]***

Exception is an unscheduled event that disrupts program execution and used to detect overflow.

Exceptions are created to handle unexpected events from within the processor.

33. How many exceptions are available in MIPS architecture?

There are two kinds of exceptions are available in MIPS architecture such as

1. Execution of an undefined instruction
2. An arithmetic overflow exception

34. What is meant by exception program counter (EPC)?

Exception program counter is a 32 bit register used to hold the address of the affected instruction.

35. What is meant by vectored interrupt?

Vectored interrupt is an interrupt for which the address to which control is transferred is determined by the cause of the exception.

36. What is meant by precise interrupt?

Precise interrupt also called as precise exception. An interrupt or exception that is always associated with the correct instruction in pipelined computers.

37. What is meant by imprecise interrupt?

Imprecise interrupt also called imprecise exception. An interrupt or exception in pipelined computers that is not associated with the extra instruction that was the cause of the interrupt or exception.

PART – B

BASIC MIPS IMPLEMENTATION

1. **Explain in detail the basic implementation of MIPS.[Nov/Dec-2015] [Page No:3.1-3.3]**
2. **Discuss in detail about the design conventions followed in MIPS processor.[Page No:3.3-3.5]**
3. **Explain in detail about the major components required to build a MIPS datapath.**

BUILDING DATAPATH

4. **Explain how various components interact together to execute different classes of instruction. .[Page No:3.5-3.8]**

[OR]

5. **Discuss how different datapath elements can be connected together to execute different formats of instructions. .[Page No:3.5-3.10]**

[OR]

6. **Explain Datapath and its controls in detail *[Nov/Dec 2014 – 16M][April/May-2018]* .[Page No:3.11-3.15]**
7. **Explain in detail the operation of the datapath. ?[Nov/Dec-2017] .[Page No:3.5-3.10]**
8. **Describe in detail how a single data path can be constructed to execute both Arithmetic logic and Memory instructions. .[Page No:3.9-3.11]**
9. **Explain the role of ALU control in overall implementation of MIPS architecture. .[Page No:3.11]**

10. **Discuss in detail the factors that are to be considered while designing the main control unit. .[Page No:3.13-3.15]**

PIPELINING

11. **Write a short note on Pipelining. .[Page No:3.15-3.20]**
12. **Explain the different types of pipeline hazards with suitable example.** ***[May/June 2015 – 16M]*** **?[Nov/Dec-2017] .[Page No:3.18-3.20]**

13. **Explain how the instruction pipeline works? What are the various situations where an instruction pipeline can stall? Illustrate with an example. .[Nov/Dec-2016] [Nov/Dec-2017] . [Page No:3.18-3.20]**

14. **Describe operand forwarding in a pipeline processor with a diagram.[April/May-2017] .[Page No:3.19]**

PIPELINE DATAPATH & CONTROL

15. **Explain how datapath is provided for a pipelined processor.** ***[April/May-2018][April/May-2016]*** **.[Page No:3.22-3.27]**
16. **Explain how control is provided for a pipelined processor.** ***[April/May-2016]*** **.[Page No:3.25-3.27]**
17. **Discuss the modified datapath to accommodate pipeline executions with a diagram.[April/May-2017] .[Page No:3.22]**

HANDLING HAZARDS

18. **Write a short note on how data hazards can be handled [*Nov/Dec 2014 – 8M]* .[Page No:3.18-3.20]**
19. **Explain the hazards caused by unconditional branching statements. .[April/May-2017-7M] .[Page No:3.28-3.30]**
20. **Why is branch prediction algorithm needed? Differentiate between the static and dynamic techniques. .[Nov/Dec-2016] .[Page No:3.31]**
21. **Write a short note on how control hazards can be handled** ***[Nov/Dec 2014 – 8M]*** **.[Page No:3.30-3.32]**
22. **Briefly explain about various categories of hazards with example.** ***[April/May-2016]*** **.[Page No:3.18-3.20]**

EXCEPTIONS

23. **Explain in detail how exceptions are handled in MIPS architecture.** ***[May/June 2015 – 16M]*** **.[Page No:3.32-3.34]**