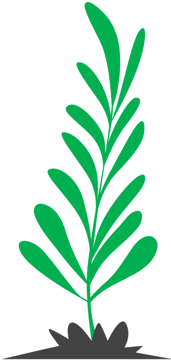
NSCET E-LEARNING PRESENTATION

LISTEN … LEARN… LEAD…

II YEAR / IV SEMESTER

CS8491-Computer Architecture



B.Sri Chithra Devi,M.E
Assistant Professor
Nadar Saraswathi College of Engineering & Technology,
Vadapudupatti, Annanji (po), Theni – 625531.

UNIT-III

PROCESSOR AND CONTROL
UNIT

PROCESSOR AND CONTROL UNIT

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

LECTURE 1-A BASIC MIPS IMPLEMENTATION:

- The implementation that includes a subset of the core MIPS instruction set:
- The memory-reference instructions **load word (lw) and store word (sw)**
- The arithmetic-logical instructions **add, sub, AND, OR, and slt**
- The instructions **branch equal (beq) and jump (j)**

An Overview of the Implementation:

For every instruction, the first two steps are identical:

1. Send the *program counter (PC) to the memory that contains the code and fetch the instruction from that memory.*
2. Read one or two registers, using fields of the instruction to select the registers to read. For the load word instruction, we need to read only one register, but most other instructions require reading two registers.

- After these two steps, the actions required to complete the instruction depend on the instruction class.
- For example, all instruction classes, except jump, use the arithmetic-logical unit (ALU) after reading the registers.
- The following diagram shows the high-level view of a MIPS implementation, focusing on the various functional units and their interconnection.

An Overview of the Implementation:

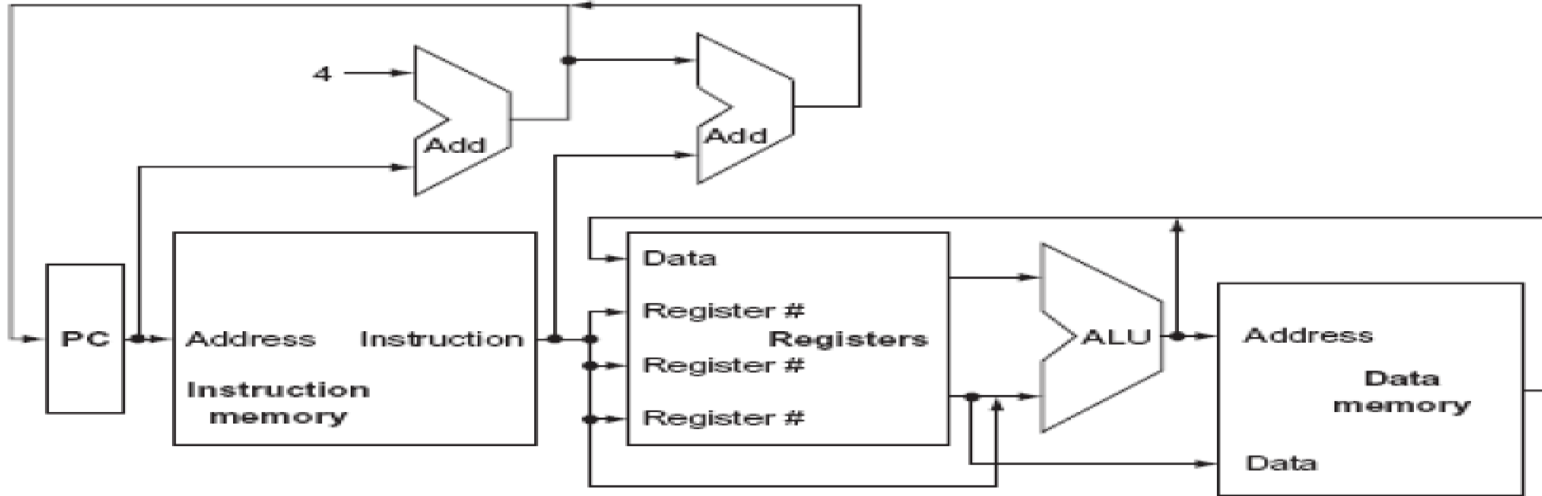


FIGURE 3.1 An abstract view of the implementation of the MIPS subset showing the major functional units and the major connections between them.

Operation

- 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 that instruction.
- Once the register operands have been fetched, all the instruction classes, except jump, use the ALU after reading the registers.
- Memory reference instructions (load or store) use the ALU for an address calculation.
- Arithmetic Logical instructions use the ALU for the operation execution.
- Branches use the ALU for comparison.

Operation

- The second input to the ALU can come from a register or the immediate field of the instruction.
- After using the ALU, the actions required to complete various instruction classes are not same.
- If the operation is a memory reference instruction a load or store, the ALU result is used as an address to either store a value from the registers or load a value from memory into the registers. The result from the ALU or memory is written back into the register file.
- If the instruction is an arithmetic-logical instruction, the result from the ALU must be written to a register.
- Branches require the use of the ALU output to determine the next instruction address, which comes either from the ALU (where the PC and branch off set are summed) or from an adder that increments the current PC by 4.

Basic implementation of MIPS with multiplexer:

- We must add a logic element that chooses from among the multiple sources and steers one of those sources to its destination. This selection is commonly done with a device called a **multiplexor, although this device might better be called a data selector which selects from among several inputs based on the setting of its control lines.**

- The control lines are set based primarily on information taken from the instruction being executed.

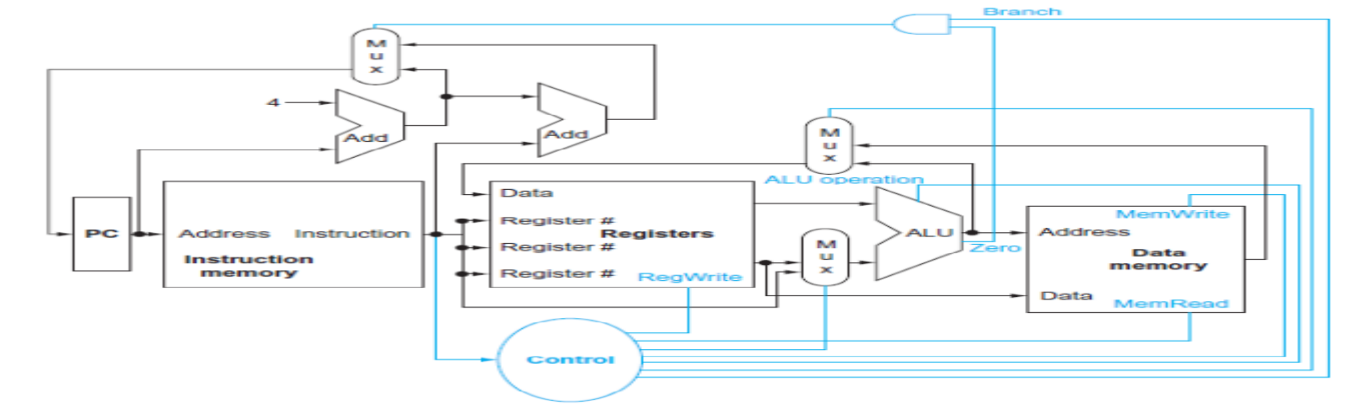
- The following figure shows the datapath with the three multiplexers added, as well as control lines for the major functional units.

Basic implementation of MIPS with multiplexer:

- A control unit is used to determine how to set the control lines for the functional units and two of the multiplexors.
- The top multiplexor ("Mux") controls what value replaces the PC (PC + 4 or the branch destination address); the multiplexor is controlled by the gate that "ANDs" together the Zero output of the ALU and a control signal that indicates that the instruction is a branch.
- The middle multiplexor, whose output returns to the register file, is used to steer the output of the ALU (in the case of an arithmetic-logical instruction) or the output of the data memory (in the case of a load) for writing into the register file.
- Finally, the bottommost multiplexor is used to determine whether the second ALU input is from the registers (for an arithmetic-logical instruction or a branch) or from the offset field of the instruction (for a load or store).

Basic implementation of MIPS with multiplexer:

- The added control lines are straightforward and determine the operation performed at the ALU, whether the data memory should read or write, and whether the registers should perform a write operation.



Logic Design Conventions:

- The datapath elements in the MIPS implementation consist of two different types of logic elements:

Combinational Elements:

- The elements that operate on data values are all **combinational, which means that their outputs depend only on the current inputs.**
- Given the same input, a combinational element always produces the same output.
- The ALU is an example of a combinational element. Given a set of inputs, it always produces the same output because it has no internal storage.

Logic Design Conventions:

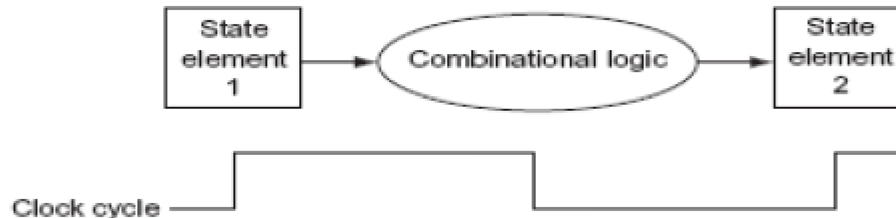
2. State Elements:

- It holds information about the state of the processor during the current clock cycle.
- An element contains state if it has some internal storage.
- All registers are state elements.
- A state element has at least two inputs and one output.
- The required inputs are the data value to be written into the element and the clock, which determines when the data value is written.
- The output from a state element provides the value that was written in an earlier clock cycle.

Logic Design Conventions:

Clocking Methodology

- A **clocking methodology defines when signals can be read and when they can be written.**
- The approach used to determine when data is valid and stable relative to the clock.
- All state elements including memory, are assumed to be positive edge-triggered; that is, they change on the rising clock edge.



Logic Design Conventions:

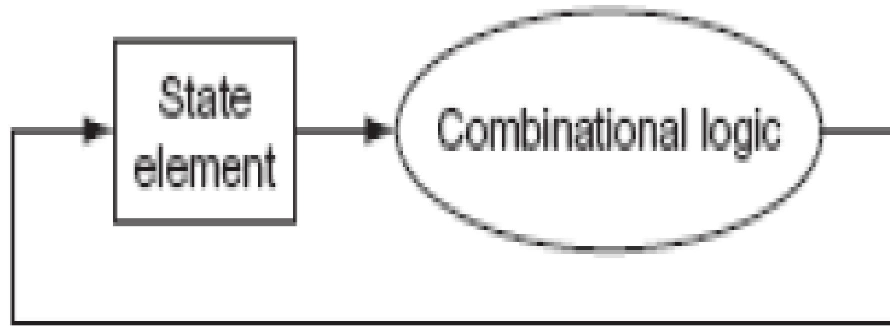
- Figure shows the two state elements surrounding a block of combinational logic, which operates in a single clock cycle.
- All signals must propagate from state element 1, through the combinational logic, and to state element 2 in the time of one clock cycle.
- The time necessary for the signals to reach state element 2 defines the length of the clock cycle.

Edge-triggered clocking methodology:

- An edge-triggered clocking methodology means that any values stored in a sequential logic element are updated only on a clock edge, which is a quick transition from low to high or vice versa.

Logic Design Conventions:

- An edge-triggered methodology allows us to read the contents of a register, send the value through some combinational logic, and write that register in the same clock cycle.



Logic Design Conventions:

Control signal

- A signal used for multiplexer selection or for directing the operation of a functional unit; contrasts with a data signal, which contains information that is operated on by a functional unit.

- **Asserted:** The signal is logically high or true.
- **Deasserted:** The signal is logically low or false.

Lecture 2-Building a datapath

Datapath

- It is a collection of function units organized in a manner to execute each class of instruction.

Datapath elements

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

.

Lecture 2-Building a datapath

How to build a datapath:

- Datapath design begins in examining the major components required to execute each class of MIPS instructions.
- First we have to know what the data path elements each instruction needs are, and also their control signals.

Stage: 1 [Datapath to fetch instruction and increment PC]

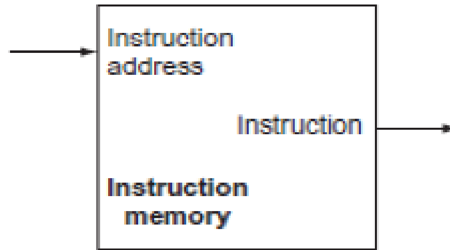
- The following diagram shows the datapath elements needed to fetch an instruction.
- The state elements are the **instruction memory, the program counter and adder.**

Nadar Saraswathi College of Engineering & Technology

Lecture 2-Building a datapath

Instruction memory

- **Instruction memory -** a memory unit to store the instructions of a program and supply instructions given an address.
- The instruction memory need only provide read access because the datapath does not write instructions.



a. Instruction memory

Lecture 2-Building a datapath

- The output at any time reflects the contents of the location specified by the address input, and no read control signal is needed.

Program counter

- The register containing the address of the instruction in the program being executed is called **program counter.**
- The program counter is a 32-bit register that is written at the end of every clock cycle and thus does not need a write control signal

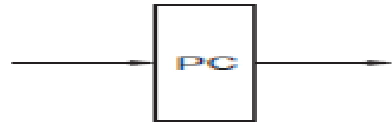
Nadar Saraswathi College of Engineering & Technology

Lecture 2-Building a datapath

- The output at any time reflects the contents of the location specified by the address input, and no read control signal is needed.

Program counter

- The register containing the address of the instruction in the program being executed is called **program counter.**
- The program counter is a 32-bit register that is written at the end of every clock cycle and thus does not need a write control signal



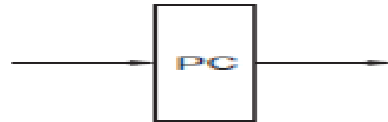
b. Program counter

Lecture 2-Building a datapath

- The output at any time reflects the contents of the location specified by the address input, and no read control signal is needed.

Program counter

- The register containing the address of the instruction in the program being executed is called **program counter.**
- The program counter is a 32-bit register that is written at the end of every clock cycle and thus does not need a write control signal



b. Program counter

Lecture 2-Building a datapath

Adder

- Adder is used to increment the PC to the address of the next instruction.

The adder is an ALU wired to always add its two 32-bit inputs and place the sum on its output.

Stage: 2 [Data path segment for multiport register file and the ALU]

Register File:

- A register file is a collection of registers in which any register can be read or written by specifying the number of the register in the file.
- R-format instructions have three register operands, so we will need to read two data words from the register file and write one data word into the register file for each instruction.

Lecture 2-Building a datapath

- For each data word to be read from the registers, we need an input to the register file that specifies the register number to be read and an output from the register file that will carry the value that has been read from the registers.
- To write a data word, we will need two inputs: one to specify the register number to be written and one to supply the data to be written into the register.
- The register number inputs are 5 bits wide to specify one of 32 registers (32 = 25), whereas the data input and two data output buses are each 32 bits wide.

Lecture 2-Building a datapath

ALU:

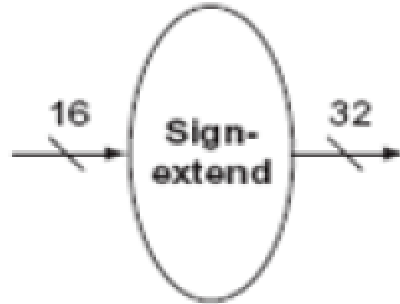
- ALU, which takes two 32-bit inputs and produces a 32-bit result, as well as a 1-bit signal if the result is 0. The inputs carrying the register number to the register file are all 5 bits wide, whereas the lines carrying data values are 32 bits wide.
- The operation to be performed by the ALU is controlled with the ALU operation signal, which will be 4 bits control signal.
- ALU provides an output signal that indicates whether the result was 0, we can send the two register operands to the ALU with the control set to do a subtract.
- If the Zero signal out of the ALU unit is asserted, we know that the two values are equal. We will be using it only to implement the equal test of branches.

Lecture 2-Building a datapath

Stage: 3 [Datapath segment for Branch Instruction]

Sign-extend

- To increase the size of a 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.



Lecture 2-Building a datapath

Branch

- A type of branch where the instruction immediately following the branch is always executed independent of whether the branch condition is true or false.

Branch taken

- A branch where the branch condition is satisfied and the program counter (PC) becomes the branch target. All unconditional jumps are taken branches.

Branch not taken or (untaken branch)

- A branch where the branch condition is false and the program counter (PC) becomes the address of the instruction that sequentially follows the branch.

Lecture 2-Building a datapath

Branch target address

- The address specified in a branch, which becomes the new program counter (PC) if the branch is taken.
- In the MIPS architecture the branch target is given by the sum of the offset field of the instruction and the address of the instruction following the branch.

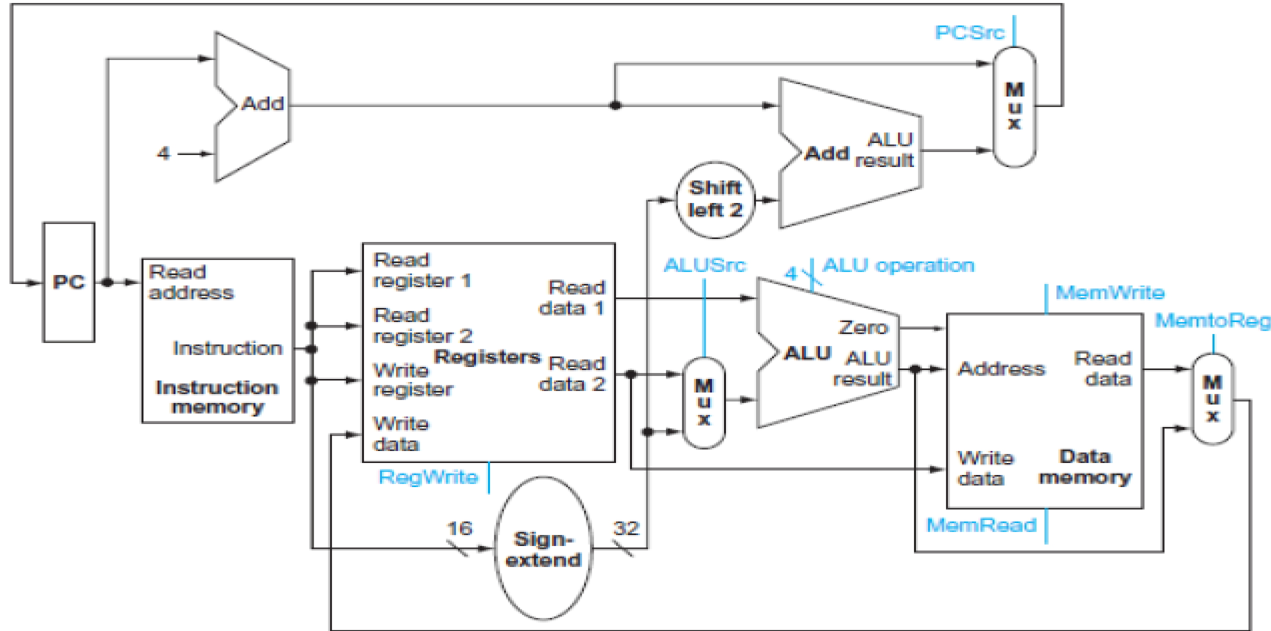
Stage: 4 [Data path Segment for Load Word and Store Word Instructions]

Data Memory

- The data memory unit is a state element with inputs for the address and the write data, and a single output for the read result. It has separate read and write controls to control the read and write operations.

Lecture 2-Building a datapath

Building a Datapath with all the stages:



Lecture 3-A CONTROL IMPLEMENTATION SCHEME

- This simple implementation covers load word (lw), store word (sw), branch equal (beq), and the arithmetic-logical instructions add, sub, AND, OR, and set on less than.

The ALU Control

- The MIPS ALU in defines the 6 following combinations of four control inputs

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

Lecture 3-A CONTROL IMPLEMENTATION SCHEME

- Depending on the instruction class, the ALU will need to perform one of these first five functions.
- For load word and store word instructions, we use the ALU to compute the memory address by addition.
- For the R-type instructions, the ALU needs to perform one of the five actions (AND, OR, subtract, add, or set on less than), depending on the value of the 6-bit funct (or function) field in the low-order bits of the instruction.
- For branch equal, the ALU must perform a subtraction.

Lecture 3-A CONTROL IMPLEMENTATION SCHEME

- We can generate the 4-bit ALU control input using a small control unit that has as inputs the function field of the instruction and a 2-bit control field, which we call ALUOp.
- The 2 bits ALUOp is interpreted as shown in Table.

ALUOp	Action
00	loads and stores
01	subtract for beq
10	determined by the operation encoded in the funct field
11	--

Lecture 3-A CONTROL IMPLEMENTATION SCHEME

- The following table shows how to set the ALU control inputs based on the 2-bit ALUOp control and the 6-bit function code.

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	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

Lecture 3-A CONTROL IMPLEMENTATION SCHEME

Designing the Main Control Unit

- Designing other controls than ALU controls begins with identifying the fields of an instruction and the control lines that are needed for the datapath.
- There are three instruction classes: the R-type, branch, and load-store instructions. The following diagram shows these formats.
- 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 the R-type instructions, branch equal, and store.
- The 16-bit off set for branch equal, load, and store is always in positions 15:0.
- The destination register is in one of two places. For a load it is in bit positions 20:16 (rt), while for an R-type instruction it is in bit positions 15:11 (rd).

Lecture 3-A CONTROL IMPLEMENTATION SCHEME

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

Single-cycle implementation:

- An implementation in which an instruction is executed in one clock cycle called single clock cycle implementation.

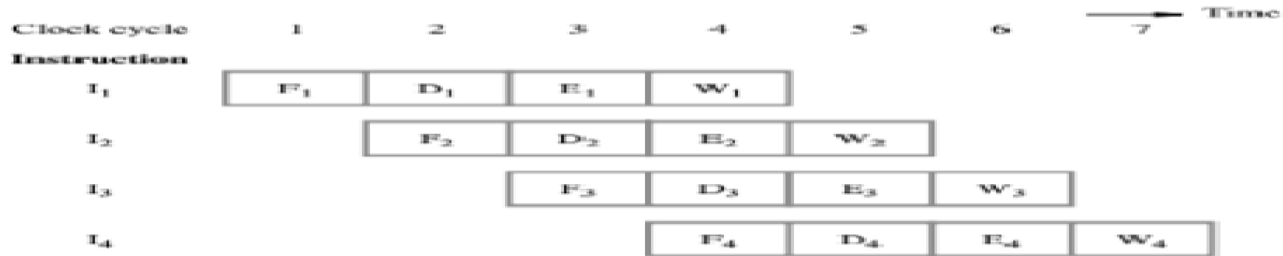
Nadar Saraswathi College of Engineering & Technology

Lecture 4-Pipelining

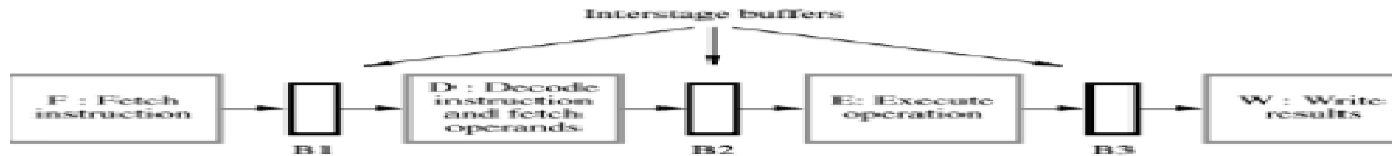
- An implementation technique in which multiple instructions are overlapped in execution is called pipeline. The different pipelining stages are,
- **Fetch** - Fetch instruction from memory.
- **Decode** - Read registers while decoding the instruction. The regular format of MIPS
- instructions allow reading and decoding to occur simultaneously.
- **Execute** - Execute the operation or calculate an address.
- **Access** - Access an operand in data memory.
- **Write** - Write the result into a register.

Lecture 4-Pipelining

Four stage Instruction Pipelining



(a) Instruction execution divided into four steps



(b) Hardware organization

Lecture 4-Pipelining

Pipeline Performance (or) Speedup

- The potential increase in performance resulting from pipelining is proportional to the number of pipeline stages.
- If all the stages take about the same amount of time and there is enough work to do, then the speed-up due to pipelining is equal to the number of stages in the pipeline.
- If the stages are perfectly balanced, then the time between instructions on the pipelined processor – assuming ideal conditions – is equal to
- $$\text{Time between instructions}_{\text{pipelined}} = \frac{\text{Time between instruction}_{\text{nonpipelined}}}{\text{Number of pipe stages}}$$

Lecture 4-Pipelining

Six stages in the pipeline:

1. Fetch instruction: Instructions are fetched from the memory into a temporary buffer before it gets executed.
2. Decode instruction: The instruction is decoded by the CPU so that the necessary op codes and operands can be determined.
3. Calculate operand: Based on the addressing scheme used, either operands are directly provided in the instruction or the effective address has to be calculated.
4. Fetch Operand: Once the address is calculated, the operands need to be fetched from the address that was calculated. This is done in this phase.
5. Execute Instruction: The instruction can now be executed.
6. Write operand: Once the instruction is executed, the result from the execution needs to be stored or written back in the memory.

Lecture 4-Pipelining

PIPELINED DATAPATH AND CONTROL

The division of an instruction into five stages means a five-stage pipeline, which in turn means that up to five instructions will be in execution during any single clock cycle.

1. IF: Instruction fetch
2. ID: Instruction decode and register file read
3. EX: Execution or address calculation
4. MEM: Data memory access
5. WB: Write back

- Each step of the instruction can be mapped onto the data path from left to right. The only exceptions are the update of the PC and the write-back step, shown in color, which sends either the ALU result or the data from memory to the left to be written into the register file.
- There are, however, two exceptions to this left -to-right flow of instructions

.

Lecture 4-Pipelining

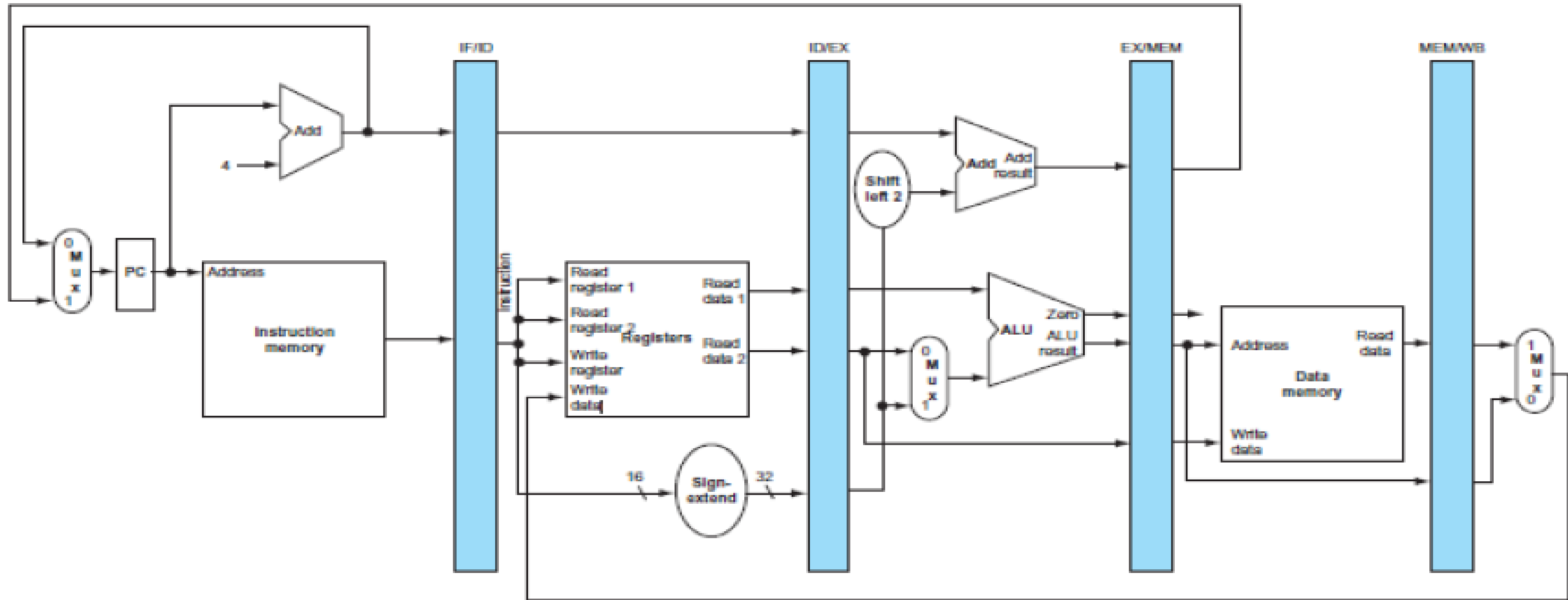
- The write-back stage, which places the result back into the register file in the middle of the datapath.
- The selection of the next value of the PC, choosing between the incremented PC and the branch address from the MEM stage Data fl owing from right to left does not affect the current instruction;
- The first right-to-left flow of data can lead to data hazards and the second leads to control hazards
- .

Lecture 4-Pipelining

The pipelined version of the datapath:

- The following diagram shows the pipelined datapath with the pipeline registers highlighted.
- 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 register between the IF and ID stages is called IF/ID.
- Notice that there is no pipeline register at the end of the write-back stage. All instructions must update some state in the processor, the register file, memory, or the PC.

Lecture 4-Pipelining



Lecture 4-Pipelining

Example: Load Instruction (lw) lw $s1, 100($s0)

1. *Instruction fetch:*

- The top portion of Figure shows the instruction being read from memory using the address in the PC and then being placed in the IF/ID pipeline register.
- The PC address is incremented by 4 and then written back into the PC to be ready for the next clock cycle.
- This incremented address is also saved in the IF/ID pipeline register in case it is needed later for an instruction, such as beq.
- The computer cannot know which type of instruction is being fetched, so it must prepare for any instruction, passing potentially needed information down the pipeline.

Lecture 4-Pipelining

2. Instruction decode and register file read:

- The bottom portion of Figure shows the instruction portion of the IF/ID pipeline register supplying the 16-bit immediate field, which is sign-extended to 32 bits, and the register numbers to read the two registers.
- All three values are stored in the ID/EX pipeline register, along with the incremented PC address.
- We again transfer everything that might be needed by any instruction during a later clock cycle.

Lecture 4-Pipelining

3. Execute or address calculation:

- The following figure shows that the load instruction reads the contents of register 1 and the sign-extended immediate from the ID/EX pipeline register and adds them using the ALU. That sum is placed in the EX/MEM pipeline register.

4. Memory access:

- The top portion of figure shows the load instruction reading the data memory using the address from the EX/MEM pipeline register and loading the data into the MEM/WB pipeline register.

Lecture 4-Pipelining

3 . Write-back:

- The bottom portion of figure shows the final step: Data memory is read using the address in the EX/MEM pipeline registers, and the data is placed in the MEM/WB pipeline register.
- Next, data is read from the MEM/WB pipeline register and written into the register file in the middle of the datapath.

Lecture 5 –Pipelined Control

PIPELINED CONTROL

- To specify control for the pipeline, we need only 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 can divide the control lines into five groups according to the pipeline stage.
- 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.
- Instruction decode/register file read: As in the previous stage, the same thing happens at every clock cycle, so there are no optional control lines to set.

Lecture 5- Pipelined Control

- Execution/address calculation: The signals to be set are RegDst, ALUOp, and ALUSrc . The signals select the Result register, the ALU operation, and either Read data 2 or a sign-extended immediate for the ALU.
- This data path borrows the control logic for PC source, register destination number, and ALU control. Note that we now need the 6-bit funct field (function code) of the instruction in the EX stage as input to ALU control, so these bits must also be included in the ID/EX pipeline register.
- Recall that these 6 bits are also the 6 least significant bits of the immediate field in the instruction, so the ID/EX pipeline register can supply them from the immediate field since sign extension leaves these bits unchanged.

Lecture 5 –Pipelined Control

- Memory access: The control lines set in this stage are Branch, MemRead, and MemWrite. The branch equal, load, and store instructions set these signals, respectively. Recall that PCSrc selects the next sequential address unless control asserts Branch and the ALU result was 0.
- Write-back: The two control lines are MemtoReg, which decides between sending the ALU result or the memory value to the register file, and Reg-Write, which writes the chosen value.
- The control lines for the final three stages. Note that four of the nine control lines are used in the EX phase, with the remaining five control lines passed on to the EX/MEM pipeline register extended to hold the control lines; three are used during the MEM stage, and the last two are passed to MEM/ WB for use in the WB stage.

Lecture 5 –Pipelined Control

- Memory access: The control lines set in this stage are Branch, MemRead, and MemWrite. The branch equal, load, and store instructions set these signals, respectively. Recall that PCSrc selects the next sequential address unless control asserts Branch and the ALU result was 0.
- Write-back: The two control lines are MemtoReg, which decides between sending the ALU result or the memory value to the register file, and Reg-Write, which writes the chosen value.
- The control lines for the final three stages. Note that four of the nine control lines are used in the EX phase, with the remaining five control lines passed on to the EX/MEM pipeline register extended to hold the control lines; three are used during the MEM stage, and the last two are passed to MEM/ WB for use in the WB stage.

Lecture 6 –Pipeline Hazards

Pipeline Hazards

- The condition that makes the pipeline to stall is called Hazards. The idle period in the pipeline execution is called Stall or Bubble.

Types of hazards:

1. Structural Hazard
2. Data Hazard
3. Control Hazard

1. Structural Hazard

- 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.

Lecture 6 –Pipeline Hazards

2. Data Hazards

- **Data hazards occur when the pipeline must be stalled because one step must wait for another to complete.**
- When a planned instruction cannot execute in the proper clock cycle because data that is needed to execute the instruction is not yet available.
- This is because of data dependence between the instructions that has been overlapped.

Lecture 6 –Pipeline Hazards

Consider the following example

add $s0, $t0, $t1

sub $t2, $s0, $t3

- In the above instruction one of the operand ($s0) of the sub instruction will be fetched only after the add instruction store it result in the same register ($s0).
- So that sub instruction is stalled for some clock cycle which makes the pipeline process to waste the some clock cycle.

Lecture 6 –Pipeline Hazards

3. Control Hazards

- It is also called branch hazard. When the proper instruction cannot execute in the proper pipeline clock cycle because the instruction that was fetched is not the one that is needed; that is, the flow of instruction addresses is not what the pipeline expected.

HANDLING DATA HAZARD:

Data hazard can be handled by using three methods.

Solution to data hazard:

1. Operand forwarding(Hardware)
2. Reordering Code (software)
3. By using stall

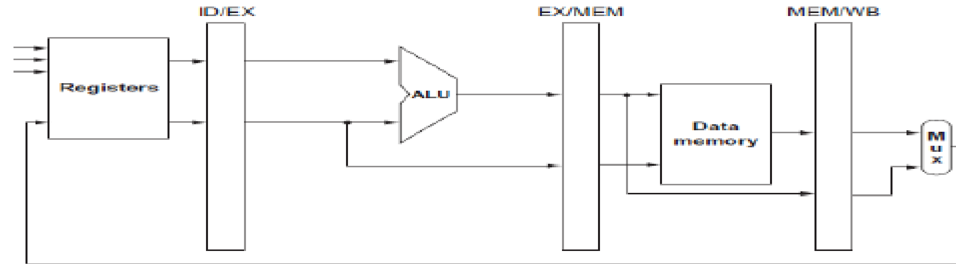
Lecture 6 –Pipeline Hazards

1. Operand forwarding (Hardware):

- 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.
- Forwarding Also called bypassing. 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.

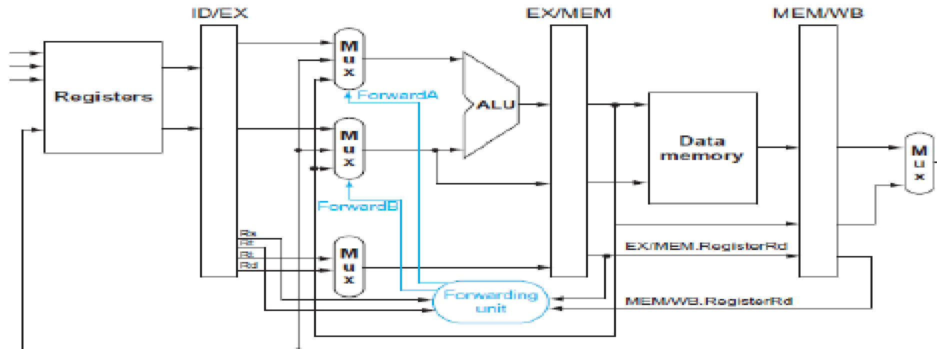
Lecture 6 –Pipeline Hazards

No forwarding:



a. No forwarding

With Forwarding:

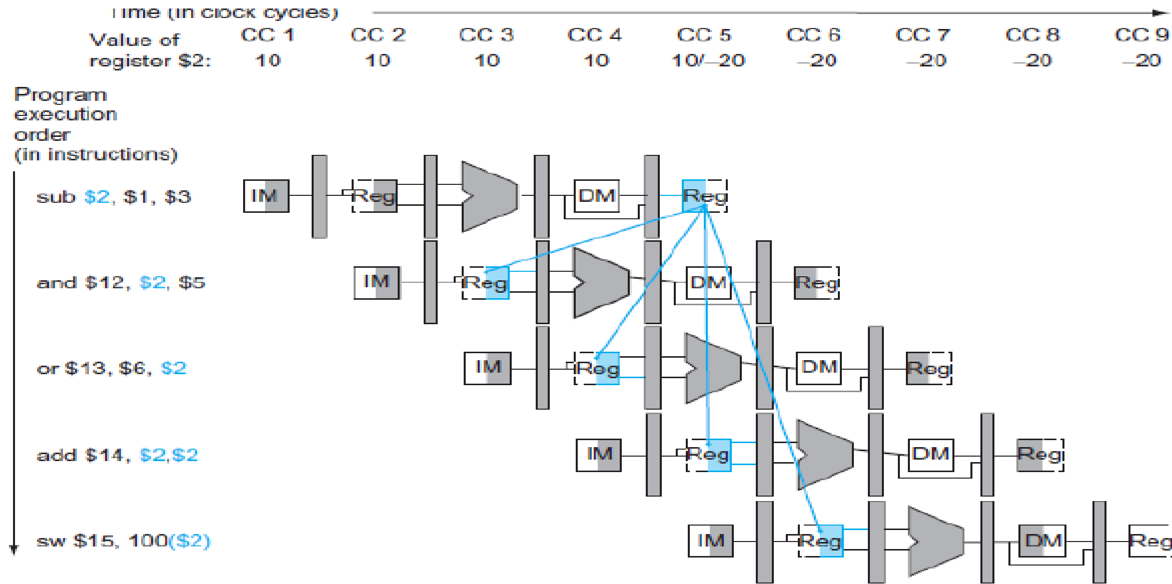


b. With forwarding

Nadar Saraswathi College of Engineering & Technology

Lecture 6 –Pipeline Hazards

Data Dependences without data forwarding Technique:



Lecture 6 –Pipeline Hazards

3. Data hazard solved by using Stall

Pipeline stall

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

Load-use data hazard

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.

nop

An instruction that does no operation to change state. **:**

Lecture 6 –Pipeline Hazards

Two schemes for resolving control hazards

1. Branch prediction
2. Delayed branching

1. Branch Prediction

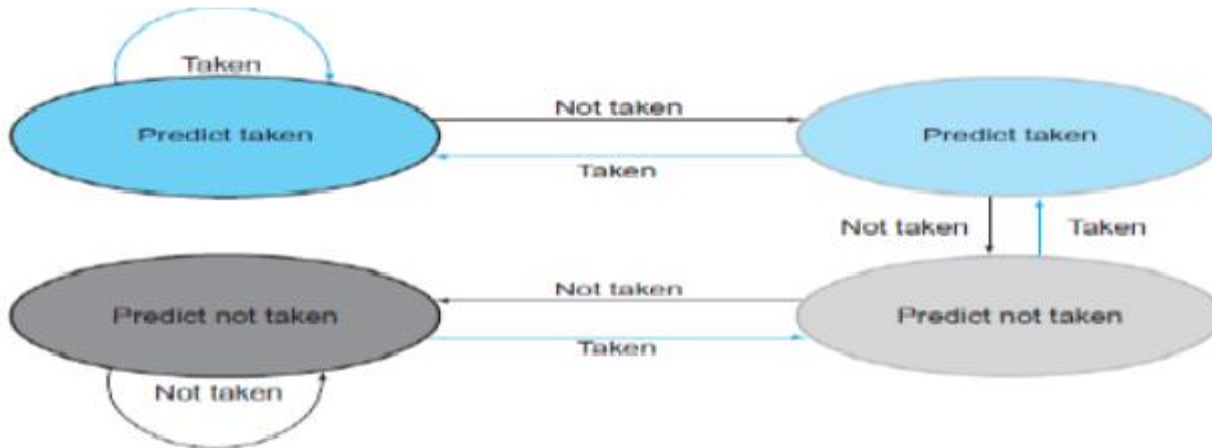
- Prediction techniques can be used to check whether a branch will be valid or not valid. These techniques reduce the branch penalty.
- A method of resolving a branch hazard that assumes a given outcome for the branch called branch prediction.
- The common prediction techniques are:
-

Lecture 6 –Pipeline Hazards

- Predict Never Taken
- Predict Always Taken
- Predict By Opcode
- Taken or Not Taken Branch
- Branch History Table
- In the first two approaches if prediction is wrong a page fault or prediction violation error occurs. The processor then halts prefetching and fetches the instruction from the desired address.
- In the third approach, the prediction is based on the opcode of the branch instruction.
- The fourth and Fifth approaches are dynamic. They depend on history of the previously executed conditional branch instruction.

Lecture 6 –Pipeline Hazards

(ii). Dynamic Branch Prediction .



Lecture 7-Exceptions

EXCEPTIONS

- Exceptions and interrupts events other than branches or jumps that change the normal flow of instruction execution.
- Exception
- Exception also called interrupt. An unscheduled event that disrupts program execution and they are used to detect overflow.
- The two types of exceptions that our current implementation can generate are execution of an undefined instruction and an arithmetic overflow.

Lecture 7-Exceptions

Interrupt

- It is an exception that comes from outside of the processor.
- We use the term interrupt only when the event is externally caused. Here are five examples showing whether the situation is internally generated by the processor or externally generated:

Handling Exception:

- The two types of exceptions can occur in the basic MIPS architecture implementation.
- Execution of an undefined instruction
- An arithmetic overflow.

Lecture 7-Exceptions

Response to an Exception:

- When an exception occurs the processor saves the address of the ending instruction in the exception program counter (EPC) and then transfer control to the operating system at some specified address.
- The operating system then takes the appropriate action, which may involve providing some service to the user program, taking some predefined action in response to an overflow, or stopping the execution of the program and reporting an error.
- After performing whatever action is required because of the exception, the operating system can terminate the program or may continue its execution, using the EPC to determine where to restart the execution of the program.

Lecture 7-Exceptions

- Two main methods used to communicate the reason for an exception:
- The first method used in the MIPS architecture is to include a status register (called the Cause register), which holds a field that indicates the reason for the exception.
- A second method is to use vectored interrupts. In a vectored interrupt, the address to which control is transferred is determined by the cause of the exception.

- For example, to accommodate the two exception types listed above, we might define the following two exception vector addresses: .

Lecture 7-Exceptions

Imprecise interrupt

- Imprecise interrupt also called imprecise exception. Interrupts or exceptions in pipelined computers that is not associated with the exact instruction that was the cause of the interrupt or exception.

Precise interrupt

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