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.

Basic MIPS Instruction

- ✓ Core MIPS instructions are divided into three classes.
  - The memory reference instructions : Load word (lw) and store word (sw)
  - The arithmetic-logic instructions : add, sub, AND, OR etc..
  - The branch instructions : branch equal (beq) and jump (j)

- ✓ For implementing every instruction, the first two steps are same:

1.Fetch the Instruction:

Send the Program Counter (PC) to the memory that contains the code and fetch the instruction from that memory.

2. Fetch the Operands :

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.
- ✓ Fortunately, for each of the three instruction classes (memory-reference, arithmetic-logical, and branches), the actions are largely the same, independent of the exact instruction.
- ✓ The simplicity and regularity of the MIPS instruction set simplifies the implementation by making the execution of many of the instruction classes similar.

- ✓ 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.
- ✓ Lastly, for a branch instruction, we may 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.

Building Datapath

The registers, ALU and interconnection bus are collectively referred to as the Datapath.

Processor

- ✓ The processing unit executed machine instructions and co-ordinates the activities of other units.
- ✓ This unit is often called Instruction Set Processor (ISP) Or simply called Processor.

FUNDAMENTAL CONCEPTS

- ✓ To execute a program, processor fetches one instruction at a time and performs the operation specified.
- ✓ Instructions are fetched from successive memory locations.

Program Counter(PC)

- ✓ The processor keeps track of the address of the memory location containing the next instruction using PC.
- ✓ After fetching an instruction the contents of the PC are updated to point to the next instruction in the sequence.

Instruction Register (IR)

- ✓ Another key register in the processor is the instruction register IR.
- ✓ Let us assume, each instruction comprises 4 bytes.

- ✓ To execute an instruction, the processor has to perform the following three steps.

  1.Fetch the content of the memory location pointed to by the PC.

  They are loaded into IR.

  That can be symbolically written as

  IR ← [[PC]]

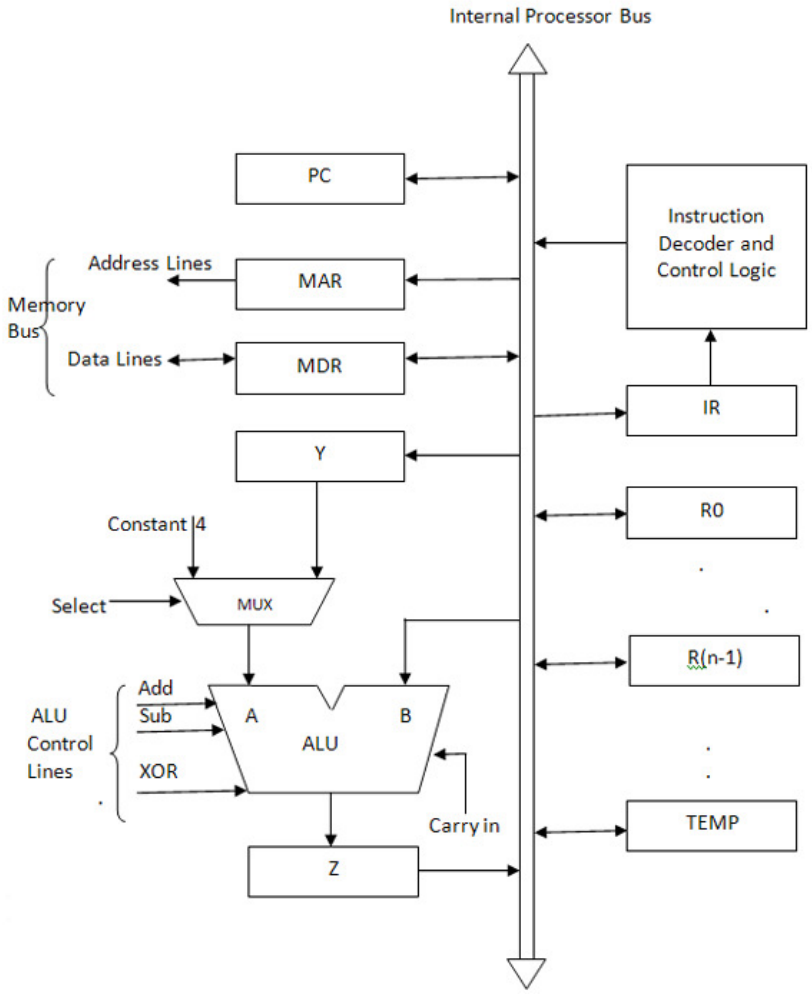
  2.Assume that the memory is byte addressable, increment the contents of the PC by 4.

  That is,

  PC ← [PC] + 4

  3.Carry out the actions specified by the instruction in the IR.

- ✓ Suppose the instruction occupies more than one word, step 1 and step 2 must be repeated as many times as necessary to fetch the complete instruction.
- ✓ These two steps are usually refereed as fetch phase.
- ✓ Step 3 constitutes the execution phase.
- ✓ The following figure shows that organization of ALU, control units and all registers using single bus.



✓ This bus is internal to the processor.

MDR

- ✓ Register MDR has two inputs and two outputs.
- ✓ Data may be loaded into MDR either from memory bus of from the internal bus.

MAR

- ✓ The input of MAR is connected to the internal bus.
- ✓ The output of MAR is connected to the external bus.

Decoder & Control logic

✓ The control lines of memory bus are connected to the instruction decoder and control logic block.

- ✓ This unit is responsible for issuing the signals that control the operation of all the units inside the processor.

General purpose registers

- ✓ The number of general purpose registers or usage of general purpose registers vary from one processor to another.
- ✓ These registers are named R0 to R(n-1)
- ✓ Some registers may be dedicated as special-purpose registers such as index registers or stack pointers.

Registers X, Y and TEMP.

- ✓ These registers are transparent to the programmer.
- ✓ That is, the programmer need not be concerned , because they are never referenced explicitly by any instruction.
- ✓ They are used by the processor temporary storage during execution of some instructions

MUX

- ✓ The multiplexer MUX selects either the output of register Y or a constant value 4 to be provided as input A of the ALU.
- ✓ The constant 4 is used to increment the contents of the program counter.
- ✓ So there are two possible values of the MUX
    1. Select4 for selecting constant 4 and
    2. SelectY for selecting Y register content.

ALU :

- ✓ It performs some arithmetic operation

Instruction Decoder & Control logic unit:

- ✓ It is responsible for implementing actions specified by the instruction loaded in the IR register.

Topic: 3

CONTROL IMPLEMENTATION SCHEME

- ✓ Consider the following instruction.

  ADD (R3) , R1

- ✓ Which adds the contents of a memory location pointed to by R3 to register R1.
- ✓ Executing this instruction requires the following actions.

  1.Fetch the instruction

  2.Fetch the first operand (memory location pointed by R3)

  3.Perform the addition.

  4.Load the result into R1.

- ✓ The following table describes the sequence of control steps, required to perform these operations.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

Step 1 : the fetch operation is initiated by loading the contents of the PC into the MAR.

And sending the Read request to the memory.

The Select signal is set to Select4, that is MUX select the constant 4.

This value is added to the operand at input B(content of PC).

The result is stored in the register Z.

Step 2 : The value in Z is moved into PC.

That is PC pointing now the next instruction.

Then waiting for memory to respond.

Step 3: The word fetched from the memory is loaded into IR.

Step 4 : the content of register R3 is transferred into MAR.

And the memory Read operation is initiated.

Step 5: The content of R1 is shifted into register Y.

Then waiting for memory to respond.

Step 6: The content of MDR is gated to bus,and register Y is selected in MUX.

Now the two values are in A and B in ALU.

Add the contents snd store it in Zin.

Step 7: The content in Z is transferred into R1.

End signal indicates the current statement is over and ready for another one.

Fetch Phase: The step 1 to 3 constitute the instruction fetch phase.

Which is same for all instructions.

Execution Phase: The step 4 to 7 constitute execution phase.

These steps are different for different instructions.

Topic: 4

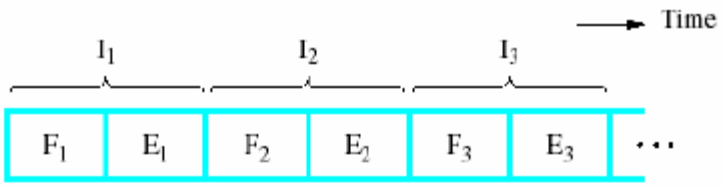
PIPELINING

- ✓ Using pipeline modern computers to achieve high performance.
- ✓ Pipelined organization requires sophisticated compilation techniques, and optimizing compilers have been developed for this purpose.
- ✓ Among other things, such compilers rearrange the sequence of operations to maximize the benefits of pipelined execution.

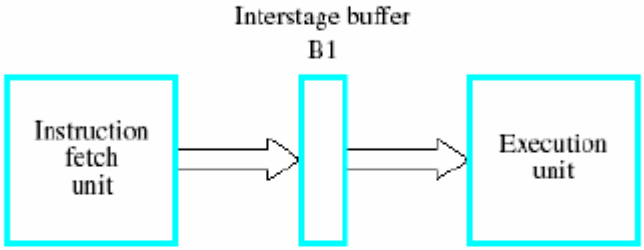
I. PIPELINE CONTROL

- ✓ The speed of execution of programs is influenced by many factors.
  - One way to improve performance is to use faster circuit technology to build the processor and the main memory.
  - Another possibility is to arrange the hardware so that more than one operation can be performed at the same time.
- ✓ In this way, the number of operations performed per second is increased even though the elapsed time needed to perform any one operation is not changed.
- ✓ The concept of multiprogramming and explained how it is possible for I/O transfers and computational activities to proceed simultaneously.

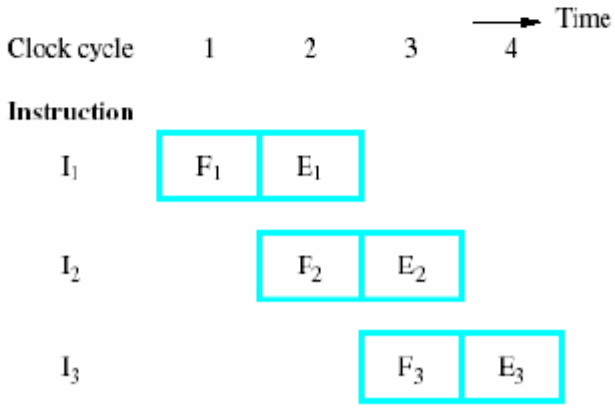
- ✓ Pipelining is a particularly effective way of organizing concurrent activity in a computer system.
- ✓ The basic idea is very simple.
- ✓ Consider how the idea of pipelining can be used in a computer.
- ✓ The processor executes a program by fetching and executing instructions, one after the other. Let Fi and Ei refer to the fetch and execute steps for instruction Ii.
- ✓ Executions of a program consist of a sequence of fetch and execute steps.
- ✓ Now consider a computer that has two separate hardware units,
    1. Fetching instructions and
    2. Executing instructions.
- ✓ The instruction is fetched by the fetch unit is deposited in an intermediate storage buffer, B1.
- ✓ This buffer is needed to enable the execution unit to execute the instruction while the fetch unit is fetching the next instruction.
- ✓ The results of execution are deposited in the destination location specified by the instruction.
- ✓ For the purposes of this discussion, we assume that both the source and the destination of the data operated on by the instructions are inside the block labeled “Execution unit.”
- ✓ The computer is controlled by a clock whose period is such that the fetch and execute steps of any instruction can each be completed in one clock cycle.
- ✓ Operation of the computer proceeds as shown in the Figure c.



(a) Sequential execution



(b) Hardware organization



(c) Pipelined execution

- ✓ In the first clock cycle, the fetch unit fetches an instruction I1 (step F1) and stores it in buffer B1 at the end of the clock cycle.
- ✓ In the second clock cycle, the instruction fetch unit proceeds with the fetch operation for instruction I2 (step F2).
- ✓ Meanwhile, the execution unit performs the operation specified by instruction I1, which is available to it in buffer B1 (step E1).
- ✓ By the end of the second clock cycle, the execution of instruction I1 is completed and instruction I2 is available.

- ✓ Instruction I2 is stored in B1, replacing I1, which is no longer needed.
- ✓ Step E2 is performed by the execution unit during the third clock cycle, while instruction I3 is being fetched by the fetch unit.
- ✓ In this manner, both the fetch and execute units are kept busy all the time.
- ✓ If the pattern in Figure © can be sustained for a long time, the completion rate of instruction execution will be twice that achievable by the sequential operation depicted in Figure (a).
- ✓ In summary, the fetch and execute units in Figure (b) constitute a two-stage pipeline in which each stage performs one step in processing an instruction.
- ✓ An interstate storage buffer, B1, is needed to hold the information being passed from one stage to the next.
- ✓ New information is loaded into this buffer at the end of each clock cycle.
- ✓ The processing of an instruction need not be divided into only two steps.
- ✓ For example, a pipelined processor may process each instruction in four steps, as follows:

F Fetch: read the instruction from the memory.
D Decode: decode the instruction and fetch the source operand(s).
E Execute: perform the operation specified by the instruction.
W Write: store the result in the destination location.

- ✓ Four instructions are in progress at any given time.
- ✓ This means that four distinct hardware units are needed.
- ✓ These units must be capable of performing their tasks simultaneously and without interfering with one another.
- ✓ Information is passed from one unit to the next through a storage buffer.
- ✓ As an instruction progresses through the pipeline, all the information needed by the stages downstream must be passed along.
- ✓ For example, during clock cycle 4, the information in the buffers is as follows:
  1. Buffer B1 holds instruction I3, which was fetched in cycle 3 and is being decoded by the instruction-decoding unit.
  2. Buffer B2 holds both the source operands for instruction I2 and the specification of the operation to be performed. This is the information produced by the decoding hardware in cycle 3. The buffer also holds the information needed for the write step of instruction I2 (stepW2). Even though it is not needed by stage E, this information must be passed on to stage W in the following clock cycle to

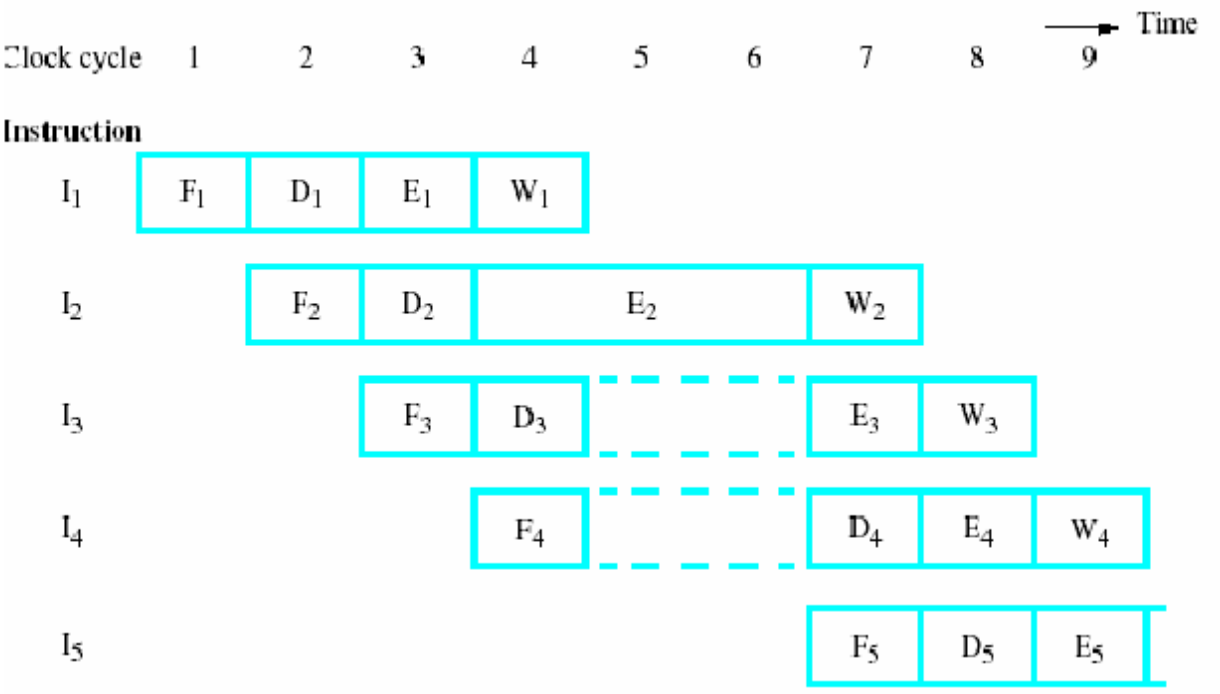
enable that stage to perform the required Write operation.

3. Buffer B3 holds the results produced by the execution unit and the destination information for instruction I1.

Topic: 5

HAZARD

- ✓ Any condition that causes the pipeline to stall is called a **hazard**.
- ✓ The below figure shows an example in which the operation specified in instruction I2 requires three cycles to complete, from cycle 4 through cycle 6.
- ✓ Thus, in cycles 5 and 6, the Write stage must be told to do nothing, because it has no data to work with.
- ✓ Meanwhile, the information in buffer B2 must remain intact until the Execute stage has completed its operation.
- ✓ This means that stage 2 and, in turn, stage 1 are blocked from accepting new instructions because the information in B1 cannot be overwritten.
- ✓ Thus, steps D4 and F5 must be postponed as shown.



- ✓ Pipelined operation is said to have been **stalled** for two clock cycles.
- ✓ This kind of stalling is called hazard.

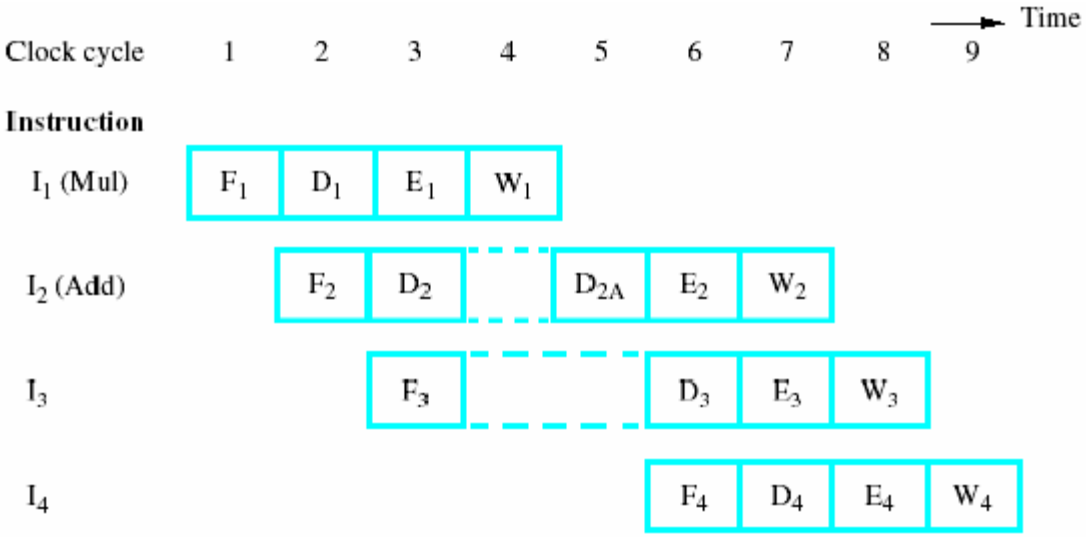
- **Types of Hazards**
  1. Data hazard
  2. Instruction or Control hazard
  3. Structure hazard

1. DATA HAZARD

- ✓ A data hazard is a situation in which the pipeline is stalled because the data to be operated on are delayed for some reason.
- ✓ We will now examine the issue of availability of data in some detail.
- ✓ Consider a program that contains two instructions, I1 followed by I2.
- ✓ When this program is executed in a pipeline, the execution of I2 can begin before the execution of I1 is completed.
- ✓ This means that the results generated by I1 may not be available for use by I2.
- ✓ The potential for obtaining incorrect results when operations are performed concurrently can be demonstrated by a simple example.
- ✓ Assume that A=5, and consider the following two operations:
  - A ← 3 + A
  - B ← 4 * A
- ✓ When these operations are performed in the order given, the result is B = 32.
- ✓ But if they are performed concurrently, the value of A used in computing B would be the
- ✓ original value, 5, leading to an incorrect result.
- ✓ If these two operations are performed by instructions in a program, then the instructions must be executed one after the other, because the data used in the second instruction depend on the result of the first instruction.
- ✓ On the other hand, the two operations
  - A← 5 * C
  - B ← 20 + C
- ✓ can be performed concurrently, because these operations are independent.
- ✓ This example illustrates a basic constraint that must be enforced to guarantee correct results.
- ✓ When two operations depend on each other, they must be performed sequentially in the correct order.

- ✓ This rather obvious condition has far-reaching consequences. Understanding its implications is the key to understanding the variety of design alternatives and trade-offs encountered in pipelined computers.

Consider the pipeline Figure.



- ✓ The data dependency just described arises when the destination of one instruction is used as a source in the next instruction.
- ✓ For example, the two instructions
  - Mul R2,R3,R4
  - Add R5,R4,R6
- ✓ give rise to a data dependency.
- ✓ The result of the multiply instruction is placed into register R4, which in turn is one of the two source operands of the Add instruction.
- ✓ Assuming that the multiply operation takes one clock cycle to complete, execution would proceed as shown in above figure.
- ✓ As the Decode unit decodes the Add instruction in cycle 3, it realizes that R4 is used as a source operand.
- ✓ Hence, the D step of that instruction cannot be completed until the W step of the multiply instruction has been completed.
- ✓ Completion of step D2 must be delayed to clock cycle 5, and is shown as step D2A in the figure.
- ✓ Instruction I3 is fetched in cycle 3, but its decoding must be delayed because step D3 cannot precede D2.

✓ Hence, pipelined execution is stalled for two cycles.

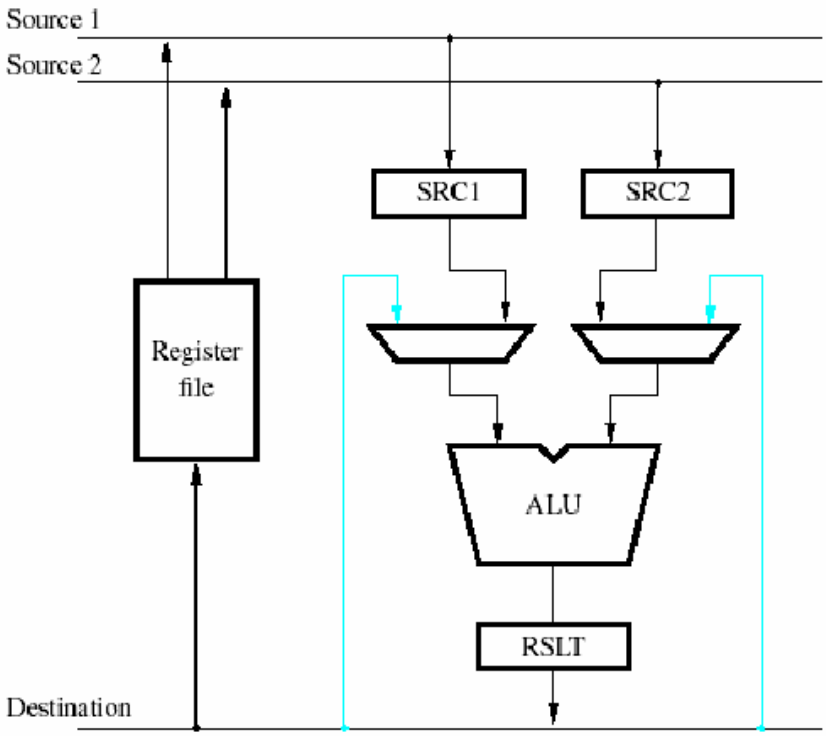
OPERAND FORWARDING

✓ The data hazard just described arises because one instruction, instruction I2 is waiting for data to be written in the register file.

✓ However, these data are available at the output of the ALU once the Execute stage completes step E1.

✓ Hence, the delay can be reduced, or possibly eliminated, if we arrange for the result of instruction I1 to be forwarded directly for use in step E2.

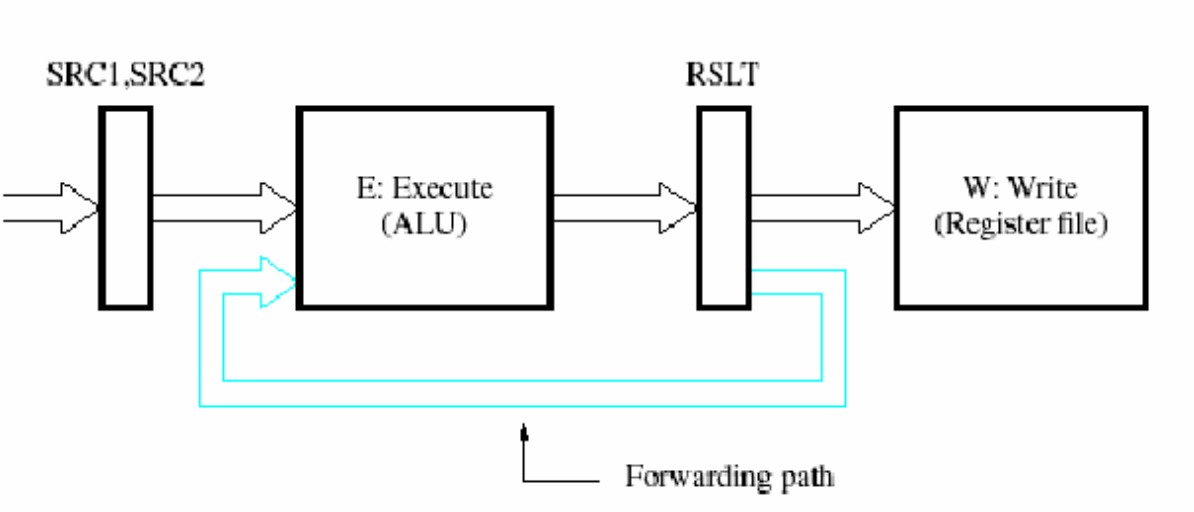
✓ The below figure shows a part of the processor data path involving the ALU and the register file.



✓ This arrangement is similar to the three-bus structure, except that registers SRC1, SRC2, and RSLT have been added.

✓ These registers constitute the inter stage buffers needed for pipelined operation in following Figure.

✓ With reference to this figure, registers SRC1 and SRC2 are part of buffer B2 and RSLT is part of B3.



- ✓ The data forwarding mechanism is provided by the blue connection lines.
- ✓ The two multiplexers connected at the inputs to the ALU allow the data on the destination bus to be selected instead of the contents of either the SRC1 or SRC2 register.
- ✓ After decoding instruction I2 and detecting the data dependency, a decision is made to use data forwarding.
- ✓ The operand not involved in the dependency, register R2, is read and loaded in register SRC1 in clock cycle 3.
- ✓ In the next clock cycle, the product produced by instruction I1 is available in register RSLT, and because of the forwarding connection, it can be used in step E2.
- ✓ Hence, execution of I2 proceeds without interruption.

HANDLING DATA HAZARDS IN SOFTWARE

- ✓ An alternative approach is to leave the task of detecting data dependencies and dealing with them to the software.
- ✓ In this case, the compiler can introduce the two-cycle delay needed between instructions I1 and I2 by inserting NOP (No-operation) instructions, as follows:

```
I1: Mul R2,R3,R4
NOP
NOP
I2: Add R5,R4,R6
```

- ✓ If the responsibility for detecting such dependencies is left entirely to the software.
- ✓ The compiler must insert the NOP instructions to obtain a correct result.

- ✓ Leaving tasks such as inserting NOP instructions to the compiler leads to simpler hardware.
- ✓ Being aware of the need for a delay, the compiler can attempt to reorder instructions to perform useful tasks in the NOP slots, and thus achieve better performance.
- ✓ On the other hand, the insertion of NOP instructions leads to larger code size.

SIDE EFFECTS

- ✓ The data dependencies encountered in the preceding examples are explicit and easily detected because the register involved is named as the destination in instruction I1and as a source in I2.
- ✓ Sometimes an instruction changes the contents of a register other than the one named as the destination.
- ✓ An instruction that uses an autoincrement or autodecrement addressing mode is an example.
- ✓ When a location other than one explicitly named in an instruction as a destination operand is affected, the instruction is said to have a side effect.
- ✓ For example, stack instructions, such as push and pop, produce similar side effects because they implicitly use the autoincrement and autodecrement addressing modes.

- ✓ Another possible side effect involves the condition code flags, which are used by instructions such as conditional branches and add-with-carry.
- ✓ Suppose that registers R1 and R2 hold a double-precision integer number that we wish to add to another double-precision number in registers R3 and R4.
- ✓ This may be accomplished as follows:

```
Add R1,R3
AddWithCarry R2,R4
```

- ✓ An implicit dependency exists between these two instructions through the carry flag.
- ✓ This flag is set by the first instruction and used in the second instruction, which performs
- ✓ the operation

R4 . [R2] + [R4] + carry

- ✓ Instructions that have side effects give rise to multiple data dependencies, which lead to a
- ✓ substantial increase in the complexity of the hardware or software needed to resolve them.

Topic: 6

2. INSTRUCTION HAZARDS

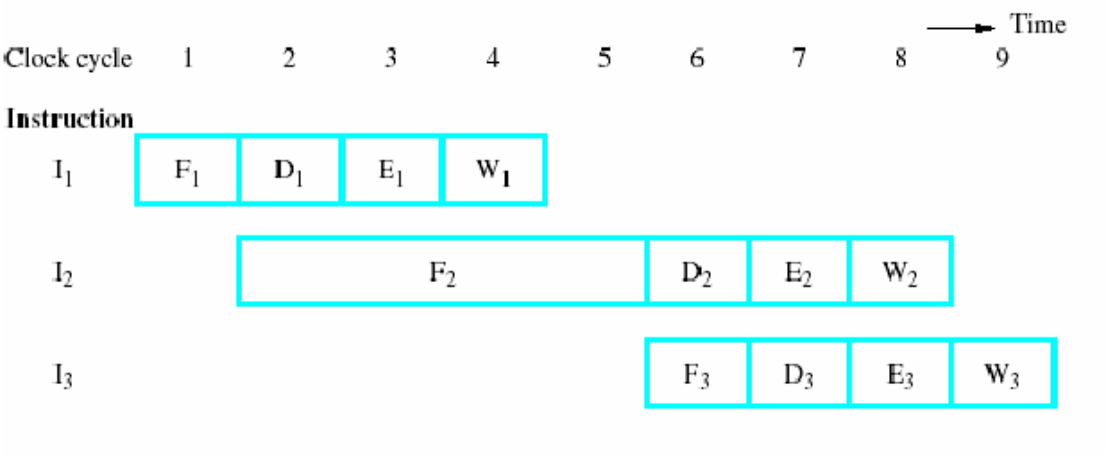
- ✓ The pipeline may also be stalled because of a delay in the availability of an instruction.
- ✓ For example, this may be a result of a miss in the cache, requiring the instruction to be fetched from the main memory.
- ✓ Such hazards are often called **control hazards or instruction hazards**.

Reasons for Instruction Hazard

- Pipeline stalls due to cache miss
- Pipeline stalls due to branch instructions

1. Pipeline stalls due to cache miss

- ✓ Instruction I1 is fetched from the cache in cycle 1, and its execution proceeds normally.
- ✓ However, the fetch operation for instruction I2, which is started in cycle 2, results in a cache miss.
- ✓ The instruction fetch unit must now suspend any further fetch requests and wait for I2 to arrive.
- ✓ We assume that instruction I2 is received and loaded into buffer B1 at the end of cycle 5.
- ✓ The pipeline resumes its normal operation at that point.



Clock cycle	1	2	3	4	5	6	7	8	9 → Time
Stage									
F: Fetch	F_1	F_2	F_2	F_2	F_2	F_3			
D: Decode		D_1	idle	idle	idle	D_2	D_3		
E: Execute			E_1	idle	idle	idle	E_2	E_3	
W: Write				W_1	idle	idle	idle	W_2	W_3

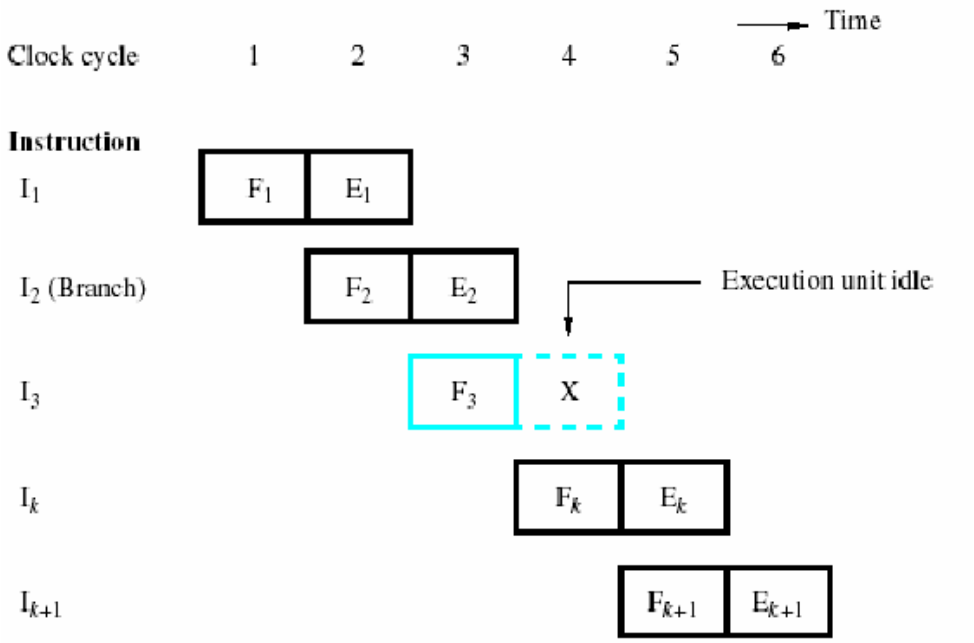
(b) Function performed by each processor stage in successive clock cycles

- ✓ An alternative representation of the operation of a pipeline in the case of a cache miss is shown in Figure b.
- ✓ This figure gives the function performed by each pipeline stage in each clock cycle.
- ✓ Note that the Decode unit is idle in cycles 3 through 5, the Execute unit is idle in cycles 4 through 6, and the Write unit is idle in cycles 5 through 7.

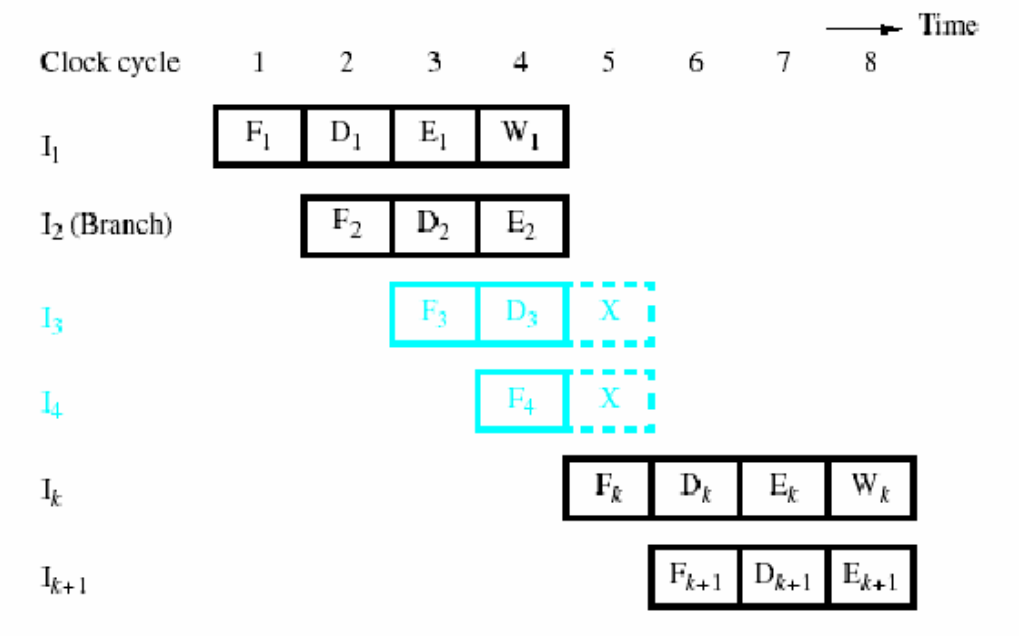
- ✓ Such idle periods are called stalls. They are also often referred to as bubbles in the pipeline.
- ✓ Once created as a result of a delay in one of the pipeline stages, a bubble moves downstream until it reaches the last unit.

2. Pipeline stalls due to branch instructions

- ✓ Assume that sequence of instructions being executed in a two-stage pipeline.
- ✓ Instructions I1 to I3 are stored at successive memory addresses, and I2 is a branch instruction.
- ✓ Let the branch target be instruction Ik .
- ✓ In clock cycle 3, the fetch operation for instruction I3 is in progress at the same time that the branch instruction is being decoded and the target address computed.
- ✓ In clock cycle 4, the processor must discard I3, which has been incorrectly fetched, and fetch instruction Ik .
- ✓ In the meantime, the hardware unit responsible for the Execute (E) step must be told to do nothing during that clock period.
- ✓ Thus, the pipeline is stalled for one clock cycle.

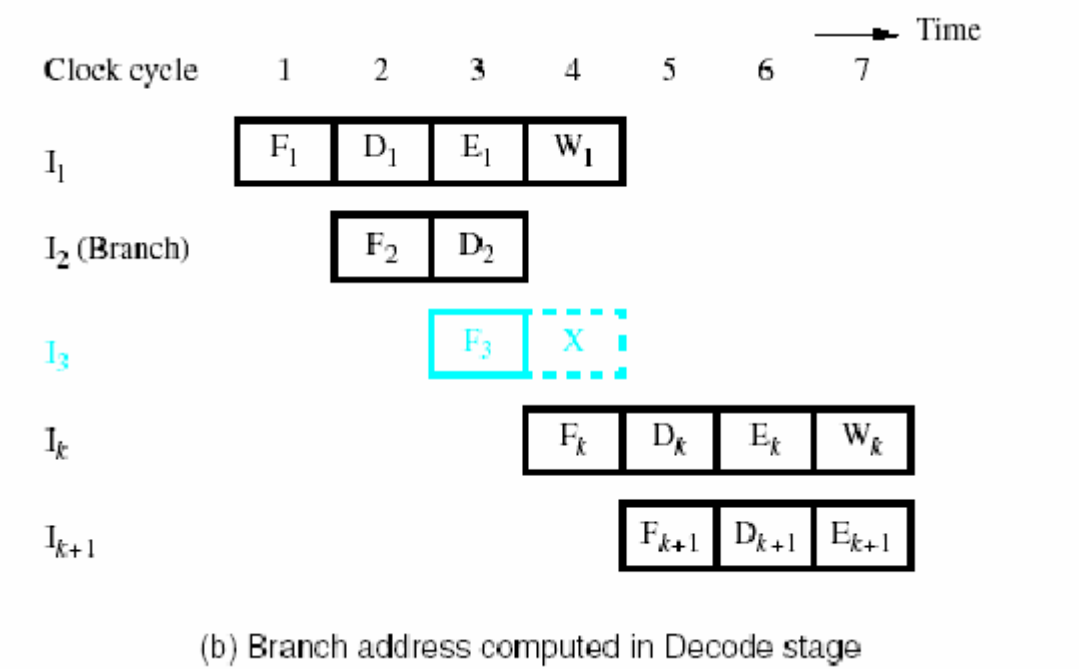


- ✓ The time lost as a result of a branch instruction is often referred to as the branch penalty.
- ✓ The branch penalty is one clock cycle.
- ✓ For a longer pipeline, the branch penalty may be higher.
- ✓ For example, following figure shows the effect of a branch instruction on a four-stage pipeline.



- ✓ We have assumed that the branch address is computed in step E2.

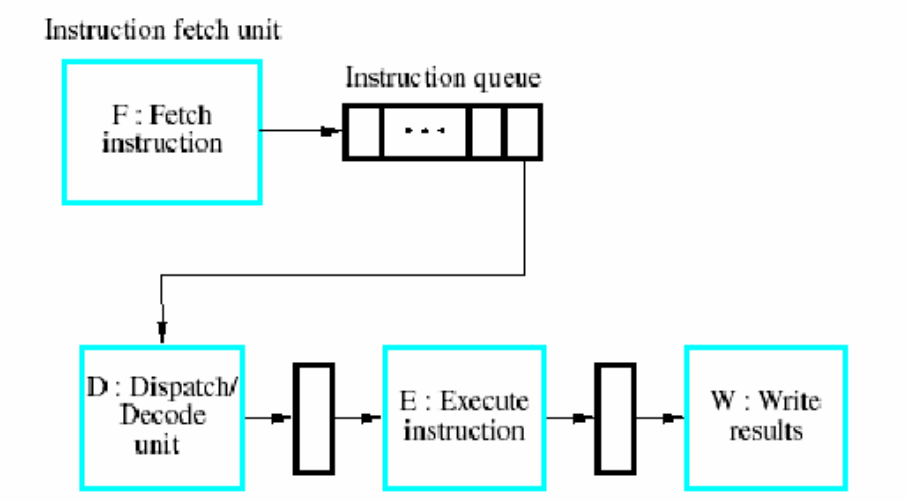
- ✓ Instructions I3 and I4 must be discarded, and the target instruction, Ik , is fetched in clock cycle 5.
- ✓ Thus, the branch penalty is two clock cycles.
- ✓ Reducing the branch penalty requires the branch address to be computed earlier in the pipeline.
- ✓ Typically, the instruction fetch unit has dedicated hardware to identify a branch instruction and compute the branch target address as quickly as possible after an instruction is fetched.
- ✓ With this additional hardware, both of these tasks can be performed in step D2, leading to the sequence of events.
- ✓ In this case, the branch penalty is only one clock cycle.



(b) Branch address computed in Decode stage

Instruction Queue and Perfecting

- ✓ Either a cache miss or a branch instruction stalls the pipeline for one or more clock cycles.
- ✓ To reduce the effect of these interruptions, many processors employ sophisticated fetch units that can fetch instructions before they are needed and put them in a queue.
- ✓ Typically, the instruction queue can store several instructions.
- ✓ A separate unit, which we call the **dispatch unit**, takes instructions from the front of the queue and sends them to the execution unit.



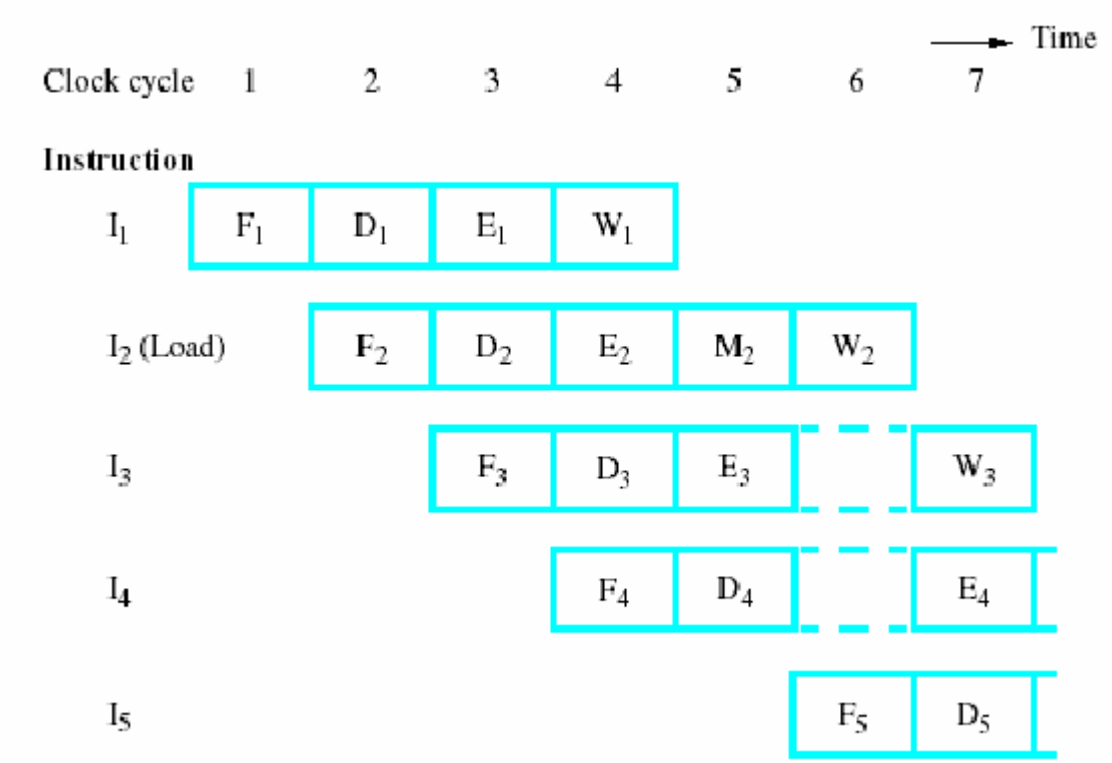
- ✓ The dispatch unit also performs the decoding function.
- ✓ To be effective, the fetch unit must have sufficient decoding and processing capability to
- ✓ recognize and execute branch instructions.
- ✓ It attempts to keep the instruction queue filled at all times to reduce the impact of occasional delays when fetching instructions.
- ✓ When the pipeline stalls because of a data hazard, for example, the dispatch unit is not able to
- ✓ issue instructions from the instruction queue.
- ✓ However, the fetch unit continues to fetch instructions and add them to the queue.
- ✓ Conversely, if there is a delay in fetching instructions because of a branch or a cache miss, the dispatch unit continues to issue instructions from the instruction queue.
- ✓ We have assumed that initially the queue contains one instruction.
- ✓ Every fetch operation adds one instruction to the queue and every dispatch operation reduces the queue length by one.

Topic: 7

3.STRUTURE HAZARD

- ✓ A third type of hazard that may be encountered in pipelined operation is known as a **structural hazard**.
- ✓ This is the situation when two instructions require the use of a given hardware resource at the same time.
- ✓ The most common case in which this hazard may arise is in access to memory.

- ✓ One instruction may need to access memory as part of the Execute or Write stage while another instruction is being fetched.
- ✓ If instructions and data reside in the same cache unit, only one instruction can proceed and the other instruction is delayed.
- ✓ Many processors use separate instruction and data caches to avoid this delay.
- ✓ An example of a structural hazard is shown in following figure.



- ✓ This figure shows how the load instruction Load X(R1),R2 can be accommodated in our example 4- stage pipeline.
- ✓ The memory address, X+[R1], is computed in stepE2 in cycle 4, then memory access takes place in cycle 5.
- ✓ The operand read from memory is written into register R2 in cycle 6.
- ✓ This means that the execution step of this instruction takes two clock cycles (cycles 4 and 5).
- ✓ It causes the pipeline to stall for one cycle, because both instructions I2 and I3 require access to the register file in cycle 6.
- ✓ Even though the instructions and their data are all available, the pipeline is stalled because one hardware resource, the register file, cannot handle two operations at once.
- ✓ If the register file had two input ports, that is, if it allowed two simultaneous write operations, the pipeline would not be stalled.

- ✓ In general, structural hazards are avoided by providing sufficient hardware resources on the processor chip.
- ✓ It is important to understand that pipelining does not result in individual instructions being executed faster;
- ✓ Rather, it is the throughput that increases, where throughput is measured by the rate at which instruction execution is completed.
- ✓ Any time one of the stages in the pipeline

Topic: 8

EXCEPTION HANDLING

- ✓ Exceptions definition: "unexpected change in control flow"
- ✓ Another form of control hazard
- ✓ There are two types of Exceptions
    - Interrupts and
    - Traps

1. Interrupts

- ✓ Caused by external events:
    - Network, Keyboard, Disk I/O, Timer
    - Page fault - virtual memory
    - System call - user request for OS action
- ✓ Asynchronous to program execution
- ✓ May be handled between instructions
- ✓ Simply suspend and resume user program

2. Traps

- Caused by internal events
    - Exceptional conditions (overflow)
    - Undefined Instruction
    - Hardware malfunction
- Usually Synchronous to program execution
- Condition must be remedied by the handler

- Instruction may be retried or program continued or program may be aborted

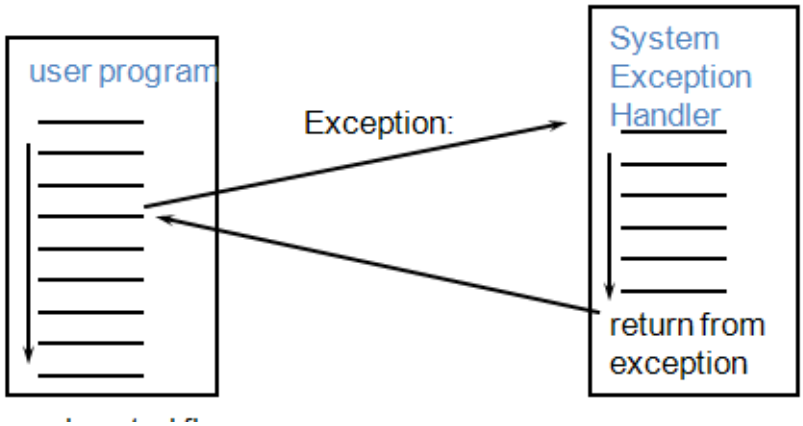
➢ The following things are happens during an exception

In The Hardware

- The pipeline has to
  1) stop executing the offending instruction in midstream,
  2) let all preceding instructions complete,
  3) flush all succeeding instructions,
  4) set a register to show the cause of the exception,
  5) save the address of the offending instruction, and
  6) then jump to a prearranged address (the address of the exception handler code)

In The Software

1) The software (OS) looks at the cause of the exception and "deals" with it.
2) Normally OS kills the program



Exception = non-programmed control transfer

- system takes action to handle the exception
- returns control to user
- must save & restore user state