BUILDING DATA PATH AND CONTROL IMPLEMENTATION SCHEME

Datapath

- Components of the processor that perform arithmetic operations and holds data.

Control

· Components ofthe processor that commands the datapath, memory, I/O devices according to the instructions of the memory.

Building a Datapath

- Elements that process data and addresses in the CPU - Memories, registers, ALUs.
- MIPS datapath can be built incrementally by considering only a subset of instructions
- 3 main elements are

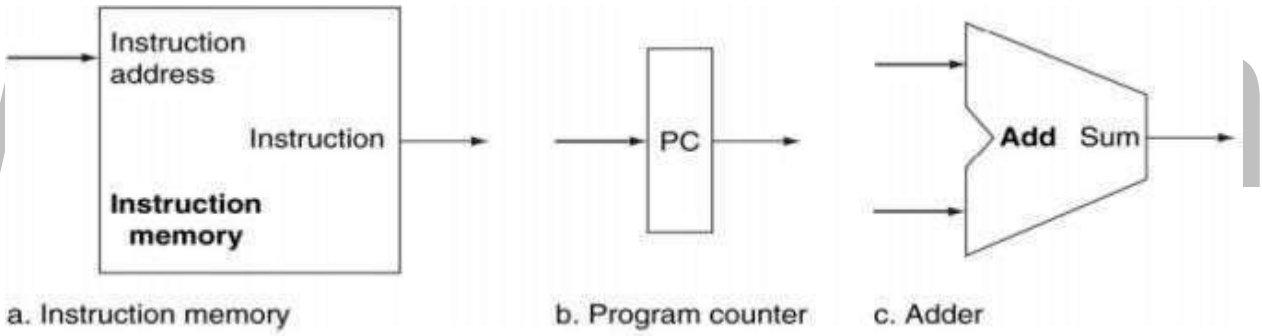

Fig. 3.2.1 Datapath

[source : V. Carl Hamacher, Zvonko G. Varanesic and Safat G. Zaky, ―Computer Organization]

A memory unit to store instructions of a program and supply instructions given an address. Needs to provide only read access (once the program is loaded).- No control signal is needed .PC (Program Counter or Instruction address register) is a register that holds the addressof the current instruction

- A new value is written to it every clock cycle. No control signal is required to enable write
- Adder to increment the PC to the address of the next instruction

An ALU permanently wired to do only addition. No extra control signal required

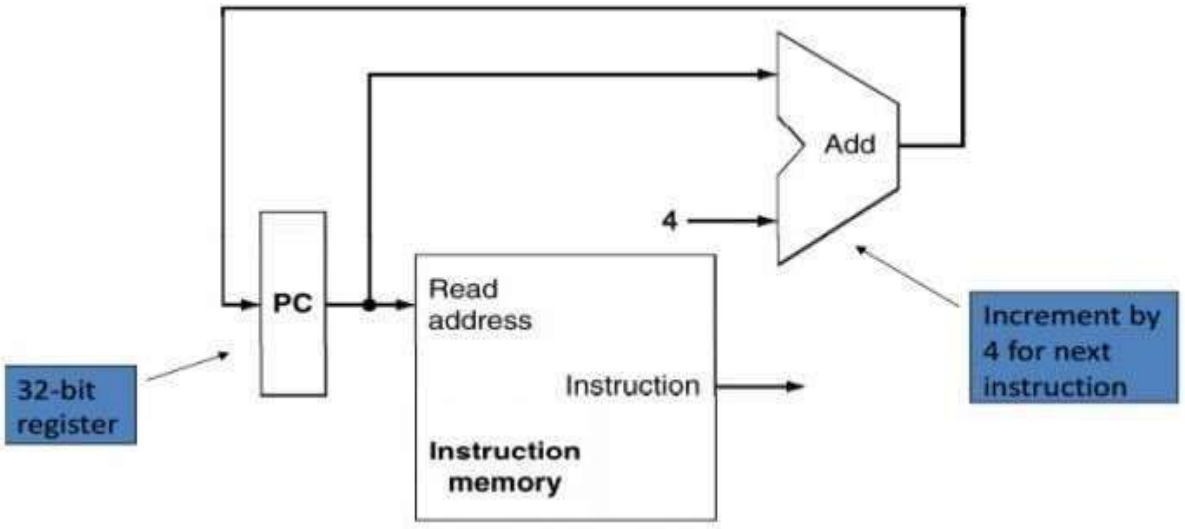

Fig. 3.2.2 Datapath portion for Instruction Fetch Types of Elements in the Datapath

[source : V. Carl Hamacher, Zvonko G. Varanesic and Safat G. Zaky, —Computer Organization]

www.binils.com

State element:

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

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

- Elements that operate on values

Eg adder ALU E.g. adder, ALU

Elements required by the different classes of instructions

- Arithmetic and logical instructions
- Data transfer instructions
- Branch instructions R-Format ALU Instructions
- E.g., add $t1, $t2, $t3

- Perform arithmetic/logical operation
- Read two register operands and write register result Register file:
- A collection of the registers
- Any register can be read or written by specifying the number of the register
- Contains the register state of the computer

Read from register

- inputs to the register file specifying the numbers
- bit wide inputs for the 32 registers
- outputs from the register file with the read values
- 32 bit wide
- For all instructions. No control required.

www.binils.com

Write to register file

- 1 input to the register file specifying the number 5 bit wide inputs for the 32 registers
- 1 input to the register file with the value to be written 32 bit wide
- Only for some instructions. RegWrite control signal.

ALU

- Takes two 32 bit input and produces a 32 bit output
- Also, sets one-bit signal if the results is 0
- The operation done by ALU is controlled by a 4 bit control signal input. This is set according to the instruction .

EXCEPTIONS

Control is the most challenging aspect of processor design: it is both the hardest part to get right and the hardest part to make fast. One of the hardest parts of control is implementing exceptions and interrupts—events other than branches or jumps that change the normal flow of instruction execution. An exception is an unexpected event from within the processor; arithmetic overflow is an example of an exception. An interrupt is an event that also causes an unexpected change in control flow but comes from outside of the processor. Interrupts are used by I/O devices to communicate with the processor.

Many architectures and authors do not distinguish between interrupts and exceptions, often using the older name interrupt to refer to both types of events. MIPS convention uses the term exception to refer to any unexpected change in control flow without distinguishing whether the cause is internal or external; we use the term interrupt only when the event is externally caused.

www.binils.com

The Intel IA-32 architecture uses the word interrupt for all these events. Interrupts were initially created to handle unexpected events like arithmetic overflow and to signal requests for service from I/O devices. The same basic mechanism was extended to handle internally generated exceptions as well. Here are some examples showing whether the situation is generated internally by the processor or externally generated:

Type of event

I/O device request - External
Invoke the operating system from user program - Internal
Arithmetic overflow - Internal
Using an undefined instruction - Internal
Hardware malfunctions - Either

The operating system knows the reason for the exception by the address at which it is initiated. The addresses are separated by 32 bytes or 8 instructions, and the operating system must record the reason for the exception and may perform some limited processing in this sequence. When the exception is not vectored, a single entry point for all exceptions can be used, and the operating system decodes the status register to find the cause.

For the operating system to handle the exception, it must know the reason for the exception, in addition to the instruction that caused it. There are two main methods used to communicate the reason for an exception. The 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.

The basic action that the processor must perform when an exception occurs is to save the address of the offending instruction in the **exception program counter (EPC)** and then transfer control to the operating system at some specified address.

www.binils.com

Interrupt & Exception

Exception Also called interrupt. An unscheduled event that disrupts program execution; used to detect overflow.

Interrupt an exception that comes from outside of the processor. (Some architectures use the term interrupt for all exceptions.)

The Intel x86 uses interrupt. We follow the MIPS convention, using the term exception. External term interrupt only when the event is externally caused. Some examples are given below in table

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

How Exceptions Are Handled in the MIPS Architecture

The basic action that the processor must perform when an exception occurs is to save the address of the offending instruction in the exception program counter (EPC) and then transfer control to the operating system at some specified address.

After performing the necessary action the operating system can either terminate the program or continue with its execution, using the EPC to determine where to restart the execution of the program.

To handle an exception, an operating system should know the reason for the exception and the instruction that caused it. Two methods are used in MIPS architecture to communicate the reason for interrupt.

1. MIPS uses a Status register or cause register which has a field that holds the reason for exception.
2. Vectored interrupts – The operating system knows the reason for the exception by the address at which it is addresses are separated by 32 bytes or eight instructions, and the initiated. The operating system must record the reason for the exception.

www.binils.com

Exceptions in a Pipelined Implementation:

A pipelined implementation treats exceptions as a form of control hazard. For example, suppose there is an arithmetic overflow in an add instruction. We must flush the instructions that follow the add instruction from the pipeline and begin fetching instructions from the new address.

The easiest way to do this is to flush the instruction and restart it from the beginning after the exception is handled. The final step is to save the address of the offending instruction in the *exception program counter* (EPC). In reality, we save the address +4, so the exception handling the software routine must first subtract 4 from the saved value.

HANDLING DATA HAZARDS & CONTROL HAZARDS

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

- Hazards reduce the performance from the ideal speedup gained by pipelining. 3 classes of hazards:
- Structural hazards: arise from resource conflicts when the hardware cannot support all possible combinations of instructions simultaneously in overlapped execution.
- Data hazards: arise when an instruction depends on the results of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline.
- Control hazards: arise from the pipelining of branches and other instructions that change the PC.

www.binils.com

Performance of Pipelines with Stalls

- A stall causes the pipeline performance to degrade from the ideal performance. Speedup from pipelining = [1/ (1+ pipeline stall cycles per instruction)] * Pipeline

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

Structural Hazards

- When a processor is pipelined, the overlapped execution of instructions requires pipelining of functional units and duplication of resources to allow all possible combinations of instructions in the pipeline.
- If some combination of instructions cannot be accommodated because of resource conflicts, the processor is said to have a structural hazard.

Instances:

- When functional unit is not fully pipelined, Then a sequence of instructions using that unpipelined unit cannot proceed at the rate of one per clock cycle.
- when some resource has not been duplicated enough to allow all combinations of instructions in the pipeline to execute.

To Resolve this hazard

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

Data Hazards

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

www.binils.com

Minimizing Data Hazard Stalls by Forwarding

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

Forwards works as:

- The ALU result from both the EX/MEM and MEM/WB pipeline registers is always fed back to the ALU inputs.
- If the forwarding hardware detects that the previous ALU operation has written the register corresponding to a source for the current ALU operation, control logic selects the forwarded result as the ALU input rather than the value read from the register file.

Data Hazards Requiring Stalls

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

Branch Hazards

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

www.binils.com

Reducing Pipeline Branch Penalties

- Simplest scheme to handle branches is to freeze or flush the pipeline, holding or deleting any instructions after the branch until the branch destination is known.
- A higher-performance, and only slightly more complex, scheme is to treat every branch as not taken, simply allowing the hardware to continue as if the branch were not executed. The complexity of this scheme arises from having to know when the state might be changed by an instruction and how to “back out” such a change.
- Implemented by continuing to fetch instructions as if the branch were a normal instruction.
- The pipeline looks as if nothing out of the ordinary is happening.
- If the branch is taken, however, we need to turn the fetched instruction into a no-op and restart the fetch at the target address.

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

Performance of Branch Schemes

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

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

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

www.binils.com

BASIC MIPS IMPLEMENTATION

1. **Instruction fetch cycle (IF):**

IR = Mem[PC];

NPC = PC + 4; Operation:

Send out the PC and fetch the instruction from memory into the instruction register (IR).

Increment the PC by 4 to address the next sequential instruction.

IR - holds instruction that will be needed on subsequent clock cycles Register NPC - holds next sequential PC.

2. **Instruction decode/register fetch cycle (ID):**

A = Regs[rs]; B = Regs[rt];

Imm = sign-extended immediate field of IR; Operation:

Decode instruction and access register file to read the registers (rs and rt -register specifiers).

Outputs of general purpose registers are read into 2 temporary registers (A and B) for use in later clock cycles.

Lower 16 bits of IR are sign extended and stored into the temporary register Imm, for use in the next cycle.

www.binils.com

3. **Execution/effective address cycle (EX):**

- ALU operates on the operands prepared in the prior cycle, performing one of four functions depending on the MIPS instruction type.

i. Memory reference: ALUOutput = A + Imm;

ii. Register-Register ALU instruction:

ALUOutput = A func B; Operation:

ALU performs the operation specified by the function code on the value in register A and in register B.

a) Result is placed in temporary register ALU Output

b) Register-Immediate ALU instruction:

ALU Output = A op Imm; Operation:

ALU performs operation specified by the opcode on the value in register A and register Imm.

Result is placed in temporary register ALU Output.

iv)Branch:

ALUOutput = NPC + (Imm << 2); Cond = (A == 0)

Operation:

- ALU adds NPC to sign-extended immediate value in Imm, which is shifted left by 2 bits to create a word offset, to compute address of branch target.
- Register A, which has been read in the prior cycle, is checked to determine whether branch is taken.

a) Considering only one form of branch (BEQZ), the comparison is against 0.

www.binils.com

4. **Memory access/branch completion cycle (MEM):**

* PC is updated for all instructions: PC = NPC; i. Memory reference: LMD = Mem[ALUOutput] or

Mem[ALUOutput] = B;

Operation:

a) Access memory if needed.

b) Instruction is load-data returns from memory and is placed in LMD (load memory data)

c) Instruction is store-data from the B register is written into memory

ii.Branch:

if (cond) PC = ALU Output

Operation: If the instruction branches, PC is replaced with the branch destination address in register ALU Output.

5. **Write-back cycle (WB):**

* Register-Register ALU instruction: Regs[rd] = ALUOutput;
* Register-Immediate ALU instruction: Regs[rt] = ALUOutput;
* Load instruction:

  Regs[rt] = LMD;

  Operation: Write the result into register file, depending on the effective opcode.

www.binils.com

PIPELINING

Basic concepts

- Pipelining is an implementation technique whereby multiple instructions are overlapped in execution
- Takes advantage of parallelism
- Key implementation technique used to make fast CPUs.
- A pipeline is like an assembly line.

In a computer pipeline, each step in the pipeline completes a part of an instruction. Like the assembly line, different steps are completing different parts of different instructions in parallel. Each of these steps is called a pipe stage or a pipe segment. The stages are connected one to the next to form a pipe—instructions enter at one end, progress through the stages, and exit at the other end.

www.binils.com

- The throughput of an instruction pipeline is determined by how often an instruction exits the pipeline.
- The time required between moving an instruction one step down the pipeline is a processor cycle..
- If the starting point is a processor that takes 1 (long) clock cycle per instruction, then pipelining decreases the clock cycle time.
- Pipeline for an integer subset of a RISC architecture that consists of load-store word, branch, and integer ALU operations.

Every instruction in this RISC subset can be implemented in at most 5 clock cycles. The 5 clock cycles are as follows.

1. **Instruction fetch cycle (IF):**

   Send the program counter (PC) to memory and fetch the current instruction from memory. PC=PC+4

2. **Instruction decode/register fetch cycle (ID):**

- Decode the instruction and read the registers.
- Do the equality test on the registers as they are read, for a possible branch.
- Compute the possible branch target address by adding the sign-extended offset to the incremented PC.

- Decoding is done in parallel with reading registers, which is possible because the register specifiers are at a fixed location in a RISC architecture, known as fixed-field decoding

3. **Execution/effective address cycle (EX):**

The ALU operates on the operands prepared in the prior cycle, performing one of three functions depending on the instruction type.

- Memory reference: The ALU adds the base register and the offset to form the effective address.

- Register-Register ALU instruction: The ALU performs the operation specified by the ALU opcode on the values read from the register file

www.binils.com

- Register-Immediate ALU instruction: The ALU performs the operation specified by the ALU opcode on the first value read from the register file and the sign-extended immediate.

4. **Memory access(MEM):**

- If the instruction is a load, memory does a read using the effective address computed in the previous cycle.

- If it is a store, then the memory writes the data from the second register read from the register file using the effective address.

5. **Write-back cycle (WB):**

Register-Register ALU instruction or Load instruction: Write the result into the register file, whether it comes from the memory system (for a load) or from the ALU (for an ALU instruction).

PIPELINED DATA PATH AND CONTROL

The Classic Five-Stage Pipeline for a RISC Processor

Each of the clock cycles from the previous section becomes a pipe stage—a cycle in the pipeline. Each instruction takes 5 clock cycles to complete, during each clock cycle the hardware will initiate a new instruction and will be executing some part of the five different instructions.

3 observations:

1. Use separate instruction and data memories, which implement with separate instruction and data caches.

2. The register file is used in the two stages: one for reading in ID and one for writing in WB, need to perform 2 reads and one write every clock cycle.

3. Does not deal with PC, To start a new instruction every clock, we must increment and store the PC every clock, and this must be done during the IF stage in preparation for the next instruction.

www.binils.com

To ensure that instructions in different stages of the pipeline do not interfere with one another. This separation is done by introducing pipeline registers between successive stages of the pipeline, so that at the end of a clock cycle all the results from a given stage are stored into a register that is used as the input to the next stage on the next clock cycle.