MODULE 5: BASIC PROCESSING UNIT

SOME FUNDAMENTAL CONCEPTS

The processing unit which executes machine instructions and coordinates the activities of other units of computer is called the Instruction Set Processor (ISP) or processor or Central Processing Unit (CPU).

The primary function of a processor is to execute the instructions stored in memory. Instructions are fetched from successive memory locations and executed in processor, until a branch instruction occurs.

• To execute an instruction, processor has to perform following 3 steps:

1. Fetch contents of memory-location pointed to by PC. Content of this location is an instruction to be executed. The instructions are loaded into IR, Symbolically, this operation is written as:
   IR ← [[PC]]
2. Increment PC by 4.
   PC ← [PC] +4
3. Carry out the actions specified by instruction (in the IR).

The steps 1 and 2 are referred to as **Fetch Phase**.
Step 3 is referred to as **Execution Phase**.

• The operation specified by an instruction can be carried out by performing one or more of the following actions:

1) Read the contents of a given memory-location and load them into a register.
2) Read data from registers or memory location.
3) Perform an arithmetic or logic operation and place the result into a register.
4) Store data from a register into a given memory-location.

• The hardware-components needed to perform these actions are shown in Figure 5.1.

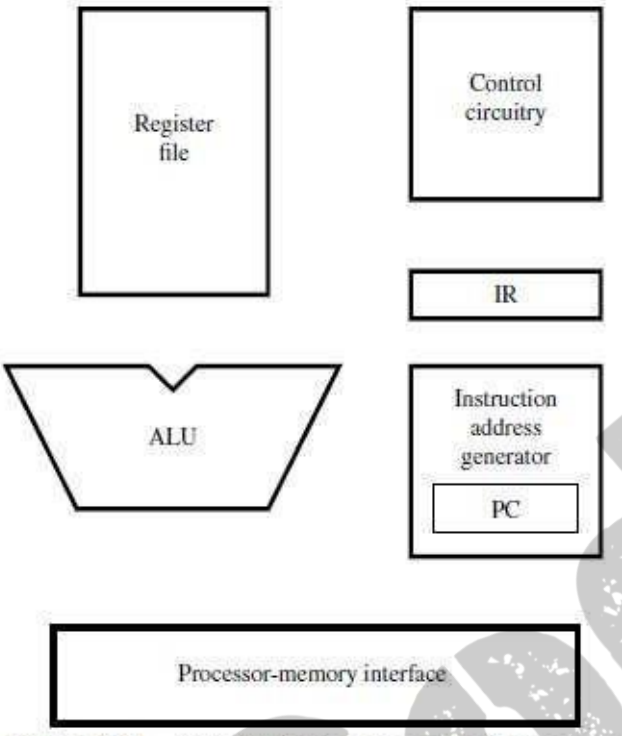


Figure 5.1 Main hardware components of a processor.

SINGLE BUS ORGANIZATION

• Here the processor contain only a single bus for the movement of data, address and instructions.
• ALU and all the registers are interconnected via a **Single Common Bus** (Figure 7.1).
• Data & address lines of the external **memory-bus** is connected to the internal processor-bus via MDR & MAR respectively.
(MDR -> Memory Data Register, MAR -> Memory Address Register).
• **MDR** has 2 inputs and 2 outputs. Data may be loaded
→ into MDR either from memory-bus (external) or
→ from processor-bus (internal).
• **MAR**'s input is connected to internal-bus; MAR's output is connected to external-bus. (address sent from processor to memory only)

• **Instruction Decoder & Control Unit** is responsible for
→ Decoding the instruction and issuing the control-signals to all the units inside the processor.
→ implementing the actions specified by the instruction (loaded in the IR).

• **Processor Registers** - Register R0 through R(n-1) are also called as General Purpose Register.
The programmer can access these registers for general-purpose use.

• **Temporary Registers** – There are 3 temporary registers in the processor. Registers - **Y**, **Z** & **Temp** are used for temporary storage during program-execution. The programmer cannot access these 3 registers.

• In **ALU**, 1) "A" input gets the operand from the output of the multiplexer (MUX).
2) "B" input gets the operand directly from the processor-bus.

• There are 2 options provided for "A" input of the ALU.

• MUX is used to select one of the 2 inputs.

• **MUX** selects either

→ output of Y or

→ constant-value 4(which is used to increment PC content).

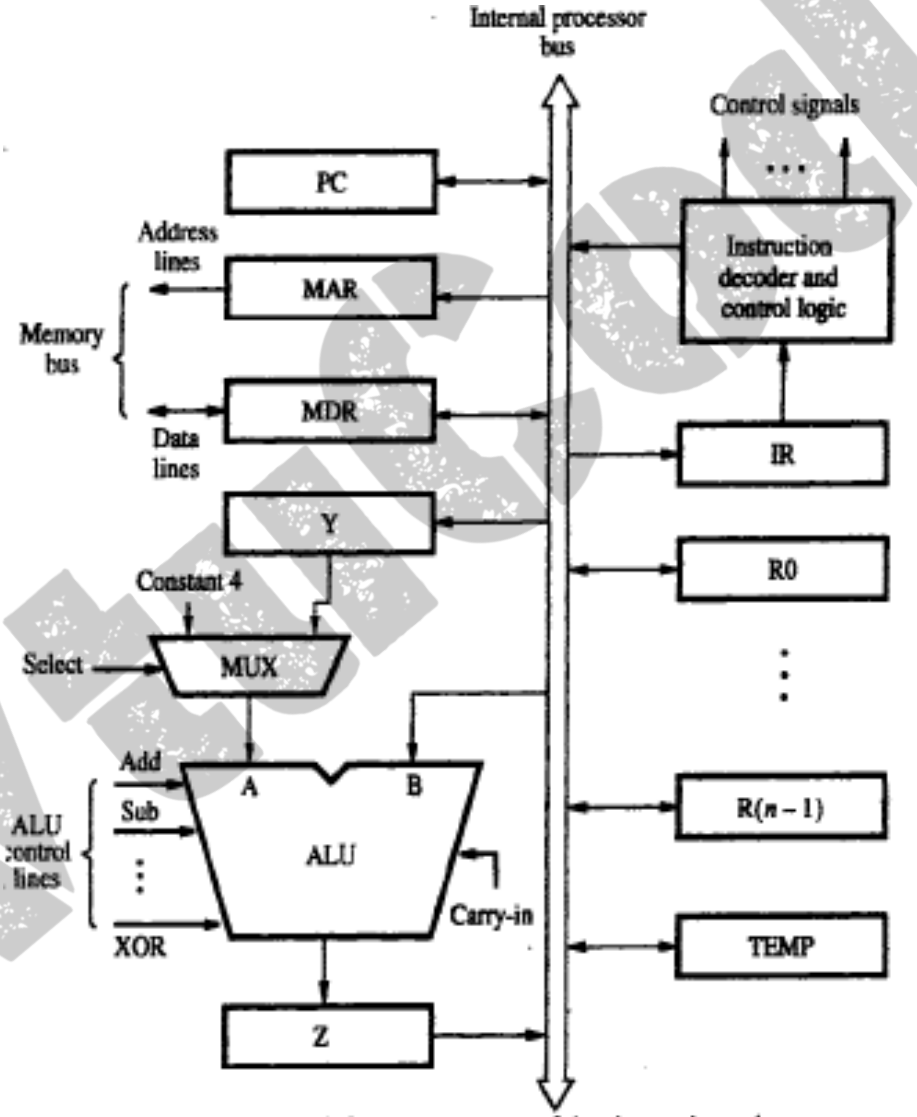


Figure 7.1 Single-bus organization of the datapath inside a processor.

• An instruction is executed by performing one or more of the following operations:

1) Transfer a word of data from one register to another or to the ALU.
2) Perform arithmetic or a logic operation and store the result in a register.

3) Fetch the contents of a given memory-location and load them into a register.
4) Store a word of data from a register into a given memory-location.

- **Disadvantage:** Only one data-word can be transferred over the bus in a clock cycle.
  Solution: Provide multiple internal-paths. Multiple paths allow several data-transfers to take place in parallel.

REGISTER TRANSFERS

- Instruction execution involves a sequence of steps in which data are transferred from one register to another.
- For each register, two control-signals are used: Riin & Riout. These are called **Gating Signals.**
- Riin=1 □ data on bus is loaded into Ri. Riout=1

□ content of Ri is placed on bus.

Riout=0, □ bus can be used for transferring data from other registers.

- For example, *Move R1, R2;* This transfers the contents of register R1 to register R2. This can be accomplished as follows:

1) Enable the output of registers R1 by setting R1out to 1 (Figure 7.2). This places the contents of R1 on processor-bus.
2) Enable the input of register R2 by setting R2out to 1.
   This loads data from processor-bus into register R4.

- All operations and data transfers within the processor take place within time-periods defined by the **processor-clock**.

• The control-signals that govern a particular transfer are asserted at the start of the clock cycle.

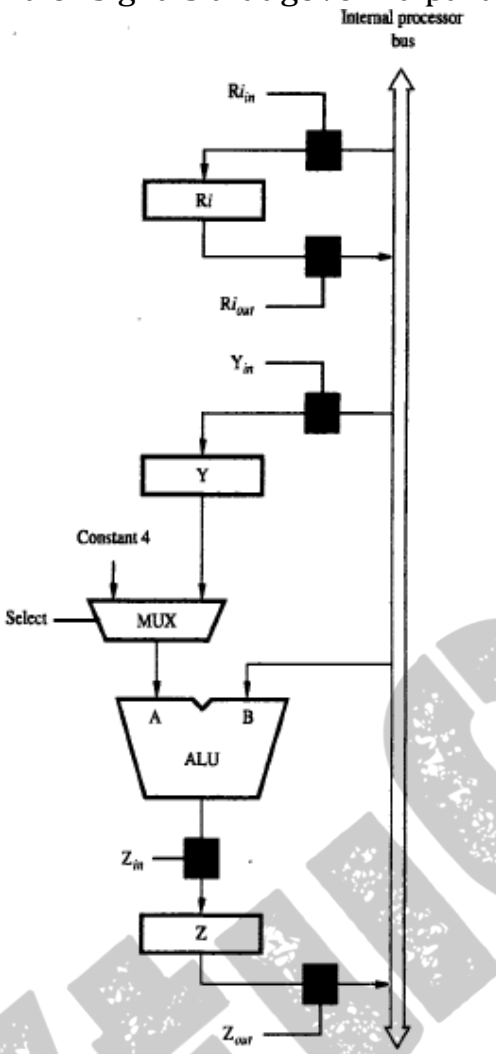


Figure 7.2 Input and output gating for the registers in Figure 7.1.

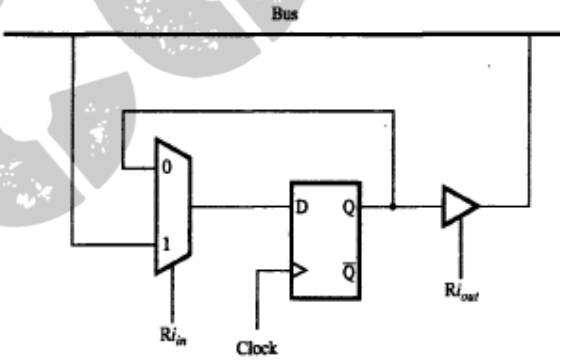


Figure 7.3 Input and output gating for one register bit.

Input & Output Gating for one Register Bit

• A 2-input multiplexer is used to select the data applied to the input of an edge-triggered D flip-flop.

- Riin=1 □ mux selects data on bus. This data will be loaded into flip-flop at rising-edge of clock. Riin=0 □ mux feeds back the value currently stored in flip-flop (Figure 7.3).
- Q output of flip-flop is connected to bus via a tri-state gate. Riout=0 □ gate's output is in the high-impedance state.

  Riout=1 □ the gate drives the bus to 0 or 1, depending on the value of Q

PERFORMING AN ARITHMETIC OR LOGIC OPERATION

• The ALU performs arithmetic operations on the 2 operands applied to its A and B inputs.

• One of the operands is output of MUX;

And, the other operand is obtained directly from processor-bus.

• The result (produced by the ALU) is stored temporarily in register Z.

• The sequence of operations for [R3] [R1]+[R2] is as follows:

1) R1out, Yin
2) R2out, SelectY, Add, Zin
3) Zout, R3in

• Instruction execution proceeds as follows:

Step 1 --> Contents from register R1 are loaded into register Y.

p2 --> Contents from Y and from register R2 are applied to the A and B inputs of ALU;

Addition is performed &

Result is stored in the Z register.

Step 3 --> The contents of Z register is stored in the R3 register.

• The signals are activated for the duration of the clock cycle corresponding to that step. All other signals are inactive.

CONTROL-SIGNALS OF MDR

• The MDR register has 4 control-signals (Figure 7.4):

1) MDRin & MDRout control the connection to the internal processor data bus &
2) MDRinE & MDRoutE control the connection to the memory Data bus.

• MAR register has 2 control-signals.

1) MARin controls the connection to the internal processor address bus &
2) MARout controls the connection to the memory address bus.

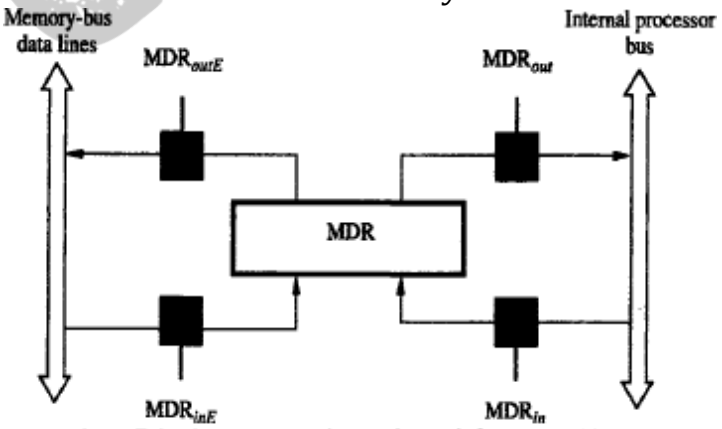


Figure 7.4 Connection and control signals for register MDR.

FETCHING A WORD FROM MEMORY

- To fetch instruction/data from memory, processor transfers required address to MAR. At the same time, processor issues Read signal on control-lines of memory-bus.
  - When requested-data are received from memory, they are stored in MDR. From MDR, they are transferred to other registers.
  - The response time of each memory access varies (based on cache miss, memory-mapped I/O). To accommodate this, MFC is used. (MFC Memory Function Completed).
  - MFC is a signal sent from addressed-device to the processor. MFC informs the processor that the requested operation has been completed by addressed-device.
  - Consider the instruction Move (R1),R2. The sequence of steps is (Figure 7.5):

1) R1out, MARin, Read ;desired address is loaded into MAR & Read command is issued.
2) MDRinE, WMFC ;load MDR from memory-bus & Wait for MFC response from memory.
3) MDRout, R2in ;load R2 from MDR.

where WMFC=control-signal that causes processor's control. circuitry to wait for arrival of MFC signal.

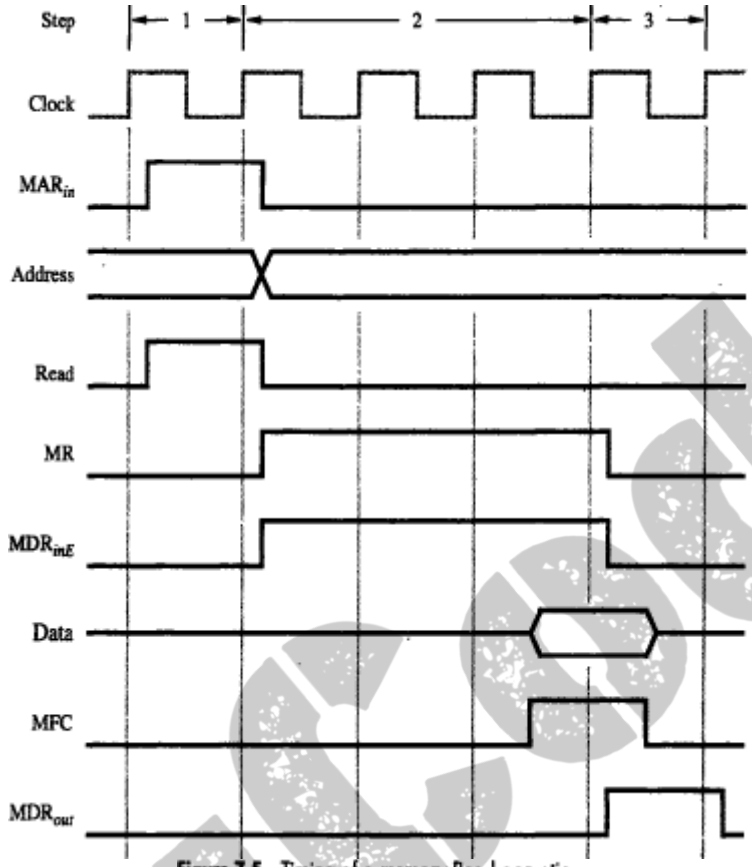


Figure 7.5 Timing of a memory Read operation.

Storing a Word in Memory

• Consider the instruction *Move R2,(R1).* This requires the following sequence:

1) R1out, MARin ;desired address is loaded into MAR.
2) R2out, MDRin, Write ;data to be written are loaded into MDR & Write command is issued.
3) MDRoutE, WMFC ;load data into memory-location pointed by R1 from MDR.

EXECUTION OF A COMPLETE INSTRUCTION

- Consider the instruction *Add (R3),R1* which adds the contents of a memory-location pointed by R3 to register R1. Executing this instruction requires the following actions:

1) Fetch the instruction.
2) Fetch the first operand.
3) Perform the addition &
4) Load the result into R1.

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

Figure 7.6 Control sequence for execution of the instruction Add (R3),R1

- Instruction execution proceeds as follows:

Step1--> The instruction-fetch operation is initiated by
→ loading contents of PC into MAR &
→ sending a Read request to memory.
The Select signal is set to Select4, which causes the Mux to select constant 4. This value is added to operand at input B (PC‟s content), and the result is stored in Z.

Step2--> Updated value in Z is moved to PC. This completes the PC increment operation and PC will now point to next instruction.

Step3--> Fetched instruction is moved into MDR and then to IR. The step 1 through 3 constitutes the **Fetch Phase**.
At the beginning of step 4, the instruction decoder interprets the contents of the IR. This enables the control circuitry to activate the control-signals for steps 4 through 7.
The step 4 through 7 constitutes the **Execution Phase**.
Step4--> Contents of R3 are loaded into MAR & a memory read signal is issued.
Step5--> Contents of R1 are transferred to Y to prepare for addition.

Step6--> When Read operation is completed, memory-operand is available in MDR, and the

addition is performed.

Step7--> Sum is stored in Z, then transferred to R1.The End signal causes a new instruction fetch cycle to begin by returning to step1.

Problem 1:

Why is the Wait-for-memory-function-completed step needed for reading from or writing to the main memory?

Solution:

The WMFC step is needed to synchronize the operation of the processor and the main memory.

Problem 2:

For the single bus organization, write the complete control sequence for the instruction: Move (R1), R1

Solution:

PCout, MARin, Read, Select4, Add, Zin

2) Zout, PCin, Yin, WMFC
3) MDRout, IRin
4) R1out, MARin, Read
5) MDRinE, WMFC
6) MDRout, R2in, End

Problem 3:

1)

Write the sequence of control steps required for the single bus organization in each of the following instructions:

a) Add the immediate number NUM to register R1.

b) Add the contents of memory-location NUM to register R1.

c) Add the contents of the memory-location whose address is at memory-location NUM to register R1.

Assume that each instruction consists of two words. The first word specifies the operation andN the addressing mode, and the second word contains the number NUM

Solution:

(*a*) 1. PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2. Z_{out}, PC_{in}, Y_{in}, WMFC
3. MDR_{out}, IR_{in}
4. PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
5. Z_{out}, PC_{in}, Y_{in}
6. $R1_{out}$, Y_{in}, WMFC
7. MDR_{out}, SelectY, Add, Z_{in}
8. Z_{out}, $R1_{in}$, End

(*b*) 1-4. Same as in (*a*)
5. Z_{out}, PC_{in}, WMFC
6. MDR_{out}, MAR_{in}, Read
7. $R1_{out}$, Y_{in}, WMFC
8. MDR_{out}, Add, Z_{in}
9. Z_{out}, $R1_{in}$, End

(*c*) 1-5. Same as in (*b*)
6. MDR_{out}, MAR_{in}, Read, WMFC
7-10. Same as 6-9 in (*b*)

Problem 4:

Show the control steps for the Branch on Negative instruction for a processor with three-bus organization of the data path

Solution:

1 PC_{out}, R=B, MAR_{in}, Read, IncPC

2 WMFC

3 MDR_{outB}, R=B, IR_{in}

4. PC_{out}, Offset field of IR_{out}, Add, If N = 1 then PC_{in}, End

5.1 Pipelining

Pipelining is a technique of decomposing a sequential process into suboperations, with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments. The name "pipeline" implies aflow of information analogous to an industrial assembly line. It is characteristic of pipelines that severalcomputations can be in progress in distinct segments at the same time.

- Perhaps the simplest way of viewing the pipeline structure is to imagine that each segment consists of an *input register* followed by a *combinational circuit*.
  - The register holds the data.
  - The combinational circuit performs the suboperation in the particular segment.
- A clock is applied to all registers after *enough time* has elapsed to perform allsegment activity.

Example

- The pipeline organization will be demonstrated by means of a simple example.
  - To perform the combined multiply and add operations with a stream ofnumbers

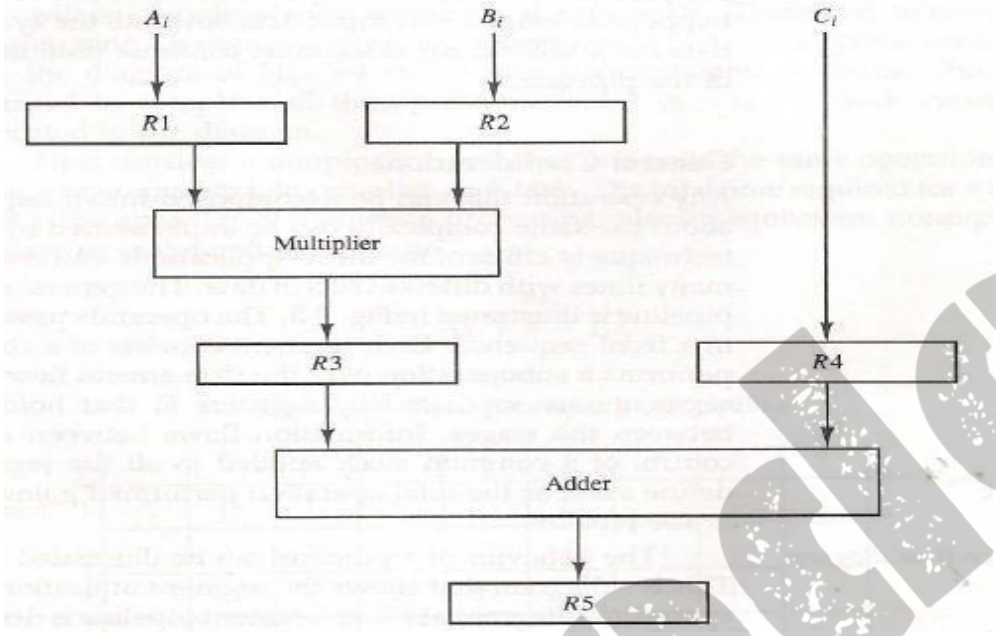
    $A_i * B_i + C_i$ for $i = 1, 2, 3, ..., 7$

- Each suboperation is to be implemented in a segment within a pipeline.
  - $R1 \leftarrow A_i$, $R2 \leftarrow B_i$ Input A_i and B_i
  - $R3 \leftarrow R1 * R2$, $R4 \leftarrow C_i$ Multiply and input C_i
  - $R5 \leftarrow R3 + R4$ Add C_i to product

The five registers are loaded with new data every clock pulse. The effect of each clock is shown in Table 9-1. The first clock pulse transfers A1 and 31 into R1 andR2. The second clockpulse transfers the product of R1 and R2 into R3 and C1into R4. The same clock pulse transfers A2 and B2 into R1 and R2. The third clock pulse operates on all three segments simultaneously. It places A3 and B3into R1 and R2, transfers the product of R1 and R2 into R3, transfers C2 intoR4, and places the sum of R3 and R4 into R5. It takes three clock pulses to fill up the pipe and retrieve the first output from R5. From there on, each clock produces a new output and moves the data one step down the pipeline. This happens as longas new input data flow into the system. When no more input data are available, the clock must continue until the last output emerges out of the pipeline.

- Each segment has one or two registers and a combinational circuit as shown in Fig. 9-2.

Figure 9-2 Example of pipeline processing.



- The five registers are loaded with new data every clock pulse. The effect of eachclock is shown in Table 9-1.

TABLE 9-1 Content of Registers in Pipeline Example

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

General considerations

- Any operation that can be decomposed into a sequence of sub operations of aboutthe *same complexity* can be implemented by a pipeline processor.
- The general structure of a four-segment pipeline is illustrated in Fig. 9-3.
- We define a *task* as the total operation performed going through all the segments inthe pipeline.
- The behavior of a pipeline can be illustrated with a *space-time* diagram.
    - It shows the segment utilization as a function of time.

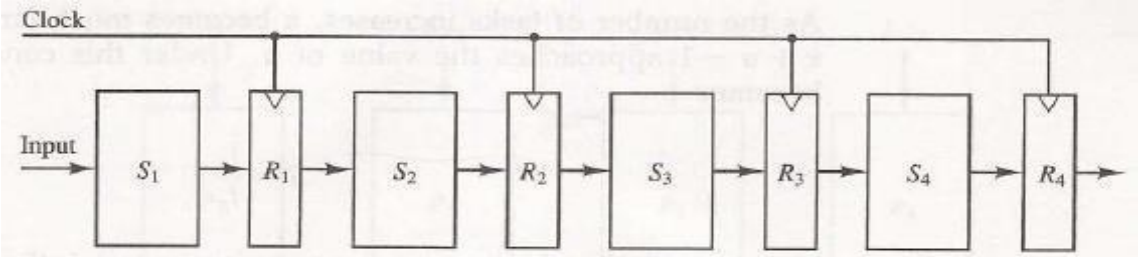


Figure 9-3 Four-segment pipeline.

- The space-time diagram of a four-segment pipeline is demonstrated in Fig. 9-4.
- Where a *k*-segment pipeline with a clock cycle time t_p is used to execute *n* tasks.
  - The first task T_1 requires a time equal to kt_p to complete its operation.
  - The remaining *n*-1 tasks will be completed after a time equal to $(n\text{-}1)t_p$
  - Therefore, to complete *n* tasks using a *k*-segment pipeline requires *k*+(*n*-1)clock cycles.
- Consider a nonpipeline unit that performs the same operation and takes a time equalto t_n to complete each task.
  - The total time required for *n* tasks is nt_n.

Figure 9-4 Space-time diagram for pipeline.

Segment:	1	2	3	4	5	6	7	8	9	Clock cycles
1	T_1	T_2	T_3	T_4	T_5	T_6				
2		T_1	T_2	T_3	T_4	T_5	T_6			
3			T_1	T_2	T_3	T_4	T_5	T_6		
4				T_1	T_2	T_3	T_4	T_5	T_6	

- The *speedup of a pipeline processing* over an equivalent nonpipeline processing isdefined by the ratio $S = nt_n/(k+n\text{-}1)t_p$.
- If *n* becomes much larger than *k*-1, the speedup becomes $S = t_n/t_p$.
- If we assume that the time it takes to process a task is the same in the pipeline andnonpipeline circuits, i.e.,$t_n = kt_p$, the speedup reduces to $S=kt_p/t_p=k$.
- This shows that the theoretical maximum speedup that a pipeline can provide is *k*,where *k* is the number of segments in the pipeline.

- To duplicate the theoretical speed advantage of a pipeline process by means of multiple functional units, it is necessary to construct *k* identical units that will beoperating in parallel.
- This is illustrated in Fig. 9-5, where four identical circuits are connected in parallel.
- Instead of operating with the *input data in sequence* as in a pipeline, the parallel circuits accept four input data items *simultaneously* and perform four tasks at thesame time.

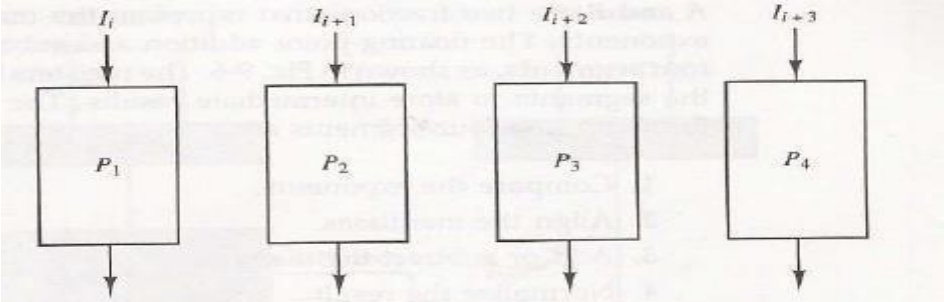


Figure 9-5 Multiple functional units in parallel.

5.2 Arithmetic Pipeline

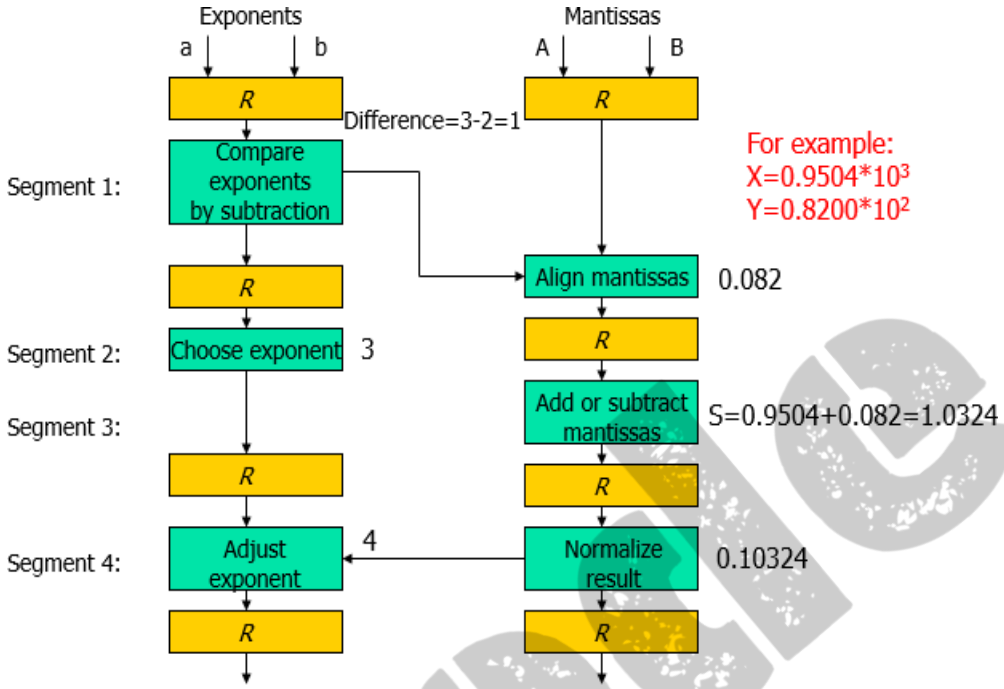
- There are various reasons why the pipeline cannot operate at its maximumtheoretical rate.
  - Different segments may take different times to complete their sub operation.
  - It is not always correct to assume that a nonpipe circuit has the same timedelay as that of an equivalent pipeline circuit.
- There are two areas of computer design where the pipeline organization isapplicable.
  - Arithmetic pipeline
  - Instruction pipeline

Arithmetic Pipeline: Introduction

- Pipeline arithmetic units are usually found in very high speed computers
  - Floating–point operations, multiplication of fixed-point numbers, and similarcomputations in scientific problem
- Floating–point operations are easily decomposed into suboperations as demonstratedin Sec. 10-5.
- An example of a pipeline unit for floating-point addition and subtraction is showed inthe following:
  - The inputs to the floating-point adder pipeline are two normalized floating-point binary number

$$X = A \times 2^a$$
$$Y = B \times 2^b$$

- A and B are two fractions that represent the mantissas, a and b are the exponents.
- The floating-point addition and subtraction can be performed in four segments, asshown in Fig. 9-6.
- The suboperations that are performed in the four segments are:
  - *Compare the exponents*
    - The larger exponent is chosen as the exponent of the result.
  - *Align the mantissas*



- The exponent difference determines how many times the mantissaassociated with the smaller exponent must be shifted to the right.

- *Add or subtract the mantissas*
- *Normalize the result*
  - When an overflow occurs, the mantissa of the sum or difference isshifted right and the exponent incremented by one.
  - If an underflow occurs, the number of leading zeros in the mantissa determines the number of left shifts in the mantissa and the numberthat must be subtracted from the exponent.

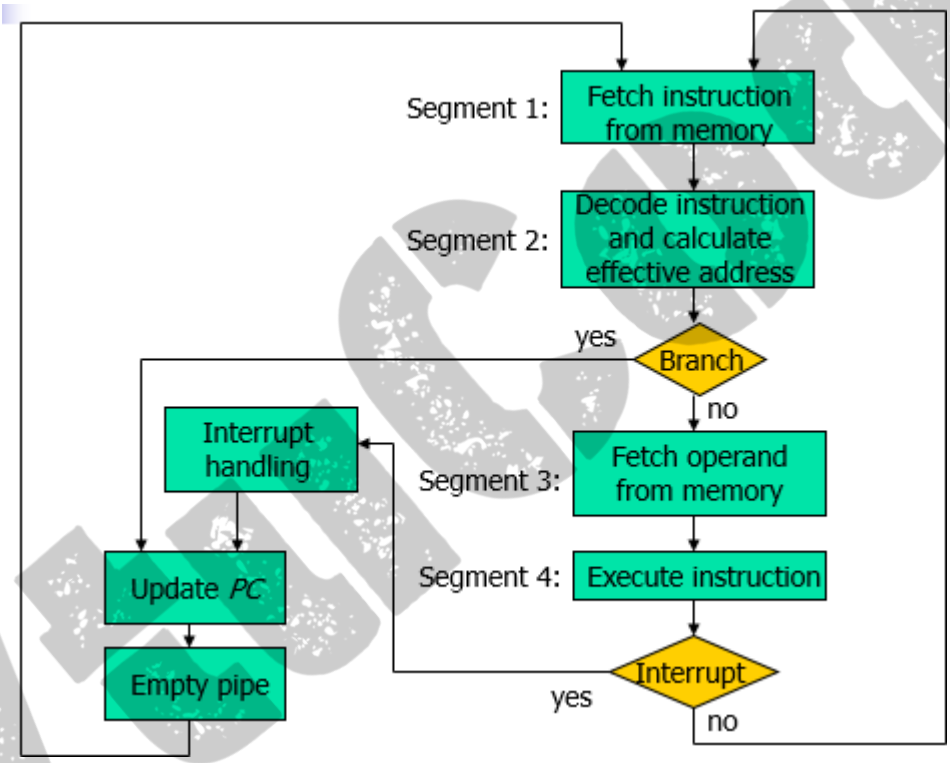
4.4 Instruction Pipeline

- Introduction:
- Pipeline processing can occur not only in the *data stream* but in the *instruction* aswell.
- Consider a computer with an instruction fetch unit (FIFO)and an instructionexecution unit(PC) designed to provide a *two-segment* pipeline.
- Computers with complex instructions require other phases in addition to abovephases to process an instruction completely.
- In the most general case, the computer needs to process each instruction with thefollowing sequence of steps.
  - Fetch the instruction from memory.
  - Decode the instruction.
  - Calculate the effective address.
  - Fetch the operands from memory.
  - Execute the instruction.
  - Store the result in the proper place.
- There are certain difficulties that will prevent the instruction pipeline from operatingat its maximum rate.

- Different segments may take different times to operate on the incominginformation.
- Some segments are skipped for certain operations.
- Two or more segments may require memory access at the same time, causing one segment to wait until another is finished with the memory.

Example: four-segment instruction pipeline:

- Assume that:
  - The decoding of the instruction can be combined with the calculation of the effective address into one segment.
  - The instruction execution and storing of the result can be combined into onesegment.
- Fig 9-7 shows how the instruction cycle in the CPU can be processed with a four- segment pipeline.
  - Thus up to four suboperations in the instruction cycle can overlap and up to four different instructions can be in progress of being processed at the sametime.



- An instruction in the sequence may be causes a branch out of normal sequence.
  - In that case the pending operations in the last two segments are completedand all information stored in the instruction buffer is deleted.
  - Similarly, an interrupt request will cause the pipeline to empty and start againfrom a new address value.
- Fig. 9-8 shows the operation of the instruction pipeline.

The time in thehorizontal axis is divided into steps of equal duration. The foursegments arerepresented in the diagram with an abbreviated symbol.

1. F1 is the segment that fetches an instruction.
2. DA is the segment that decodes the instruction and calculates theeffective address.
3. Fo is the segment that fetches the operand.
4. EX is the segment that executes the instruction.

	Step:	1	2	3	4	5	6	7	8	9	10	11	12	13
	1	FI	DA	FO	EX									
Instruction:	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	—	—	FI	DA	FO	EX			
	5					—	—	—	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

FI: the segment that fetches an instruction
DA: the segment that decodes the instruction and calculate the effective address
FO: the segment that fetches the operand
EX: the segment that executes the instruction

It is assumed that the processor has separate instruction and data memoriesso that the operation in F1 and PC can proceed at the same time. In the absenceof a branch instruction, each segment operates on different instructions. Thus,in step 4, instruction
1 is being executed in segment EX; the operand forinstruction 2 is being fetched in segment FO; instruction 3 is being decoded insegment DA; and instruction 4 is being fetched from memory in segment FI.

Assume now that instruction 3 is a branch instruction. As soon as thisinstruction is decoded in segment DA in step 4, the transfer from F1 to DA ofthe other instructions is halted until the branch instruction is executed in step6. If the branch is taken, a new instruction is fetched in step 7. If the branchis not taken, the instruction fetched previously in step 4 can be used. Thepipeline then continues until a new branch instruction is encountered.

Another delay may occur in the pipeline if the EX segment needs to storethe result of the operation in the data memory while the FO segment needsto fetch an operand. In that case, segment FO must wait until segment EX hasfinished its operation.

- In general, there are three major difficulties that cause the instruction pipeline todeviate from its normal operation.
  - *Resource conflicts* caused by access to memory by two segments at the sametime.
    - Can be resolved by using separate instruction and data memories
  - *Data dependency conflicts* arise when an instruction depends on the result ofa previous instruction, but this result is not yet available.
  - *Branch difficulties* arise from branch and other instructions that change thevalue of PC.