MODULE 5: BASIC PROCESSING UNIT

Q.1 With a figure explain single bus organization of datapath inside a processor.(8M)
Q.2 Explain how read/fetch operation from memory takes place (or) Give control sequence for Move (R1),R2
Q.3 What are actions required to execute a complete instruction Add(R3),R4. Give the control sequence for execution of instruction Add(R3),R4
Q.4 Draw and explain multiple bus organization of CPU. And write control sequence for the instruction Add R4,R5,R6.
Q.5 Explain Hardwired approach for generating control signals
Q.6Explain with a neat diagram micro programmed control method for design of control unit and write the micro routine for the instruction Branch<0
Q.7 Briefly explain the block diagram of Microwave oven
Q.8 With a block diagram explain Digital Camera
Q.9 With a block diagram explain Parallel I/O Interface
Q.10 With a block diagram explain Serial I/O Interface
Q.11With a block diagram explain Counter/Timer
Q.12 Explain multiprocessor. Justify how delay is reduced.

SOME FUNDAMENTAL CONCEPTS

- 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 can be written as
     IR□ [[PC]]
  2) Increment PC by 4
     PC□ [PC] +4
  3) Carry out the actions specified by instruction (in the IR).
- The first 2 steps are referred to as fetch phase;
  Step 3 is referred to as execution phase.

SINGLE BUS ORGANIZATION

Q.1 With a figure explain single bus organization of datapath inside a processor.(8M)

- 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, and
  MAR‟s output is connected to external-bus.
- Instruction-decoder & control-unit is responsible for

→ issuing the signals that control the operation of all the units inside the processor (and for interacting with memory bus).

→ implementing the actions specified by the instruction (loaded in the IR)

• Registers R0 through R(n-1) are provided for general purpose use by programmer.

• Three registers Y, Z & TEMP are used by processor for temporary storage during execution of some instructions. These are transparent to the programmer i.e. programmer need not be concerned with them because they are never referenced explicitly by any instruction.

• MUX(Multiplexer) selects either

→ output of Y or

→ constant-value 4(is used to increment PC content).This is provided as input A of ALU.

• B input of ALU is obtained directly from processor-bus.

• As instruction execution progresses, data are transferred from one register to another, often passing through ALU to perform arithmetic or logic operation.

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

1) Transfer a word of data from one processor-register to another or to the ALU.
2) Perform arithmetic or a logic operation and store the result in a processor-register.
3) Fetch the contents of a given memory-location and load them into a processor-register.
4) Store a word of data from a processor-register into a given memory-location.

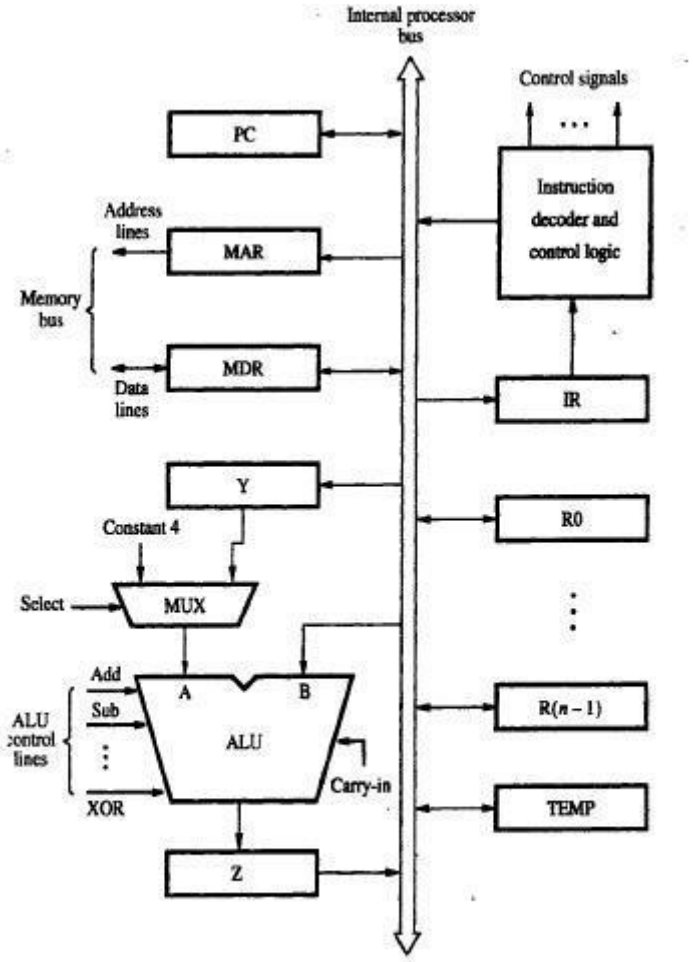


Figure 7.1 Single-bus organization of the datapath inside a processor.

REGISTER TRANSFERS

• Instruction execution involves a sequence of steps in which data are transferred from one register to another.

• Input & output of register Ri is connected to bus via switches controlled by 2 control-signals: Riin & Riout. These are called *gating signals.*

• When Riin=1, data on bus is loaded into Ri.

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

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

• When edge-triggered flip-flops are not used, 2 or more clock-signals may be needed to guarantee proper transfer of data. This is known as *multiphase clocking*.

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.

• When Riin=1, mux selects data on bus. This data will be loaded into flip-flop at rising-edge of clock.

• Q output of flip-flop is connected to bus via a tri-state gate.

When Riout=0, gate's output is in the high-impedance state. (This corresponds to the open-circuit state of a switch).

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

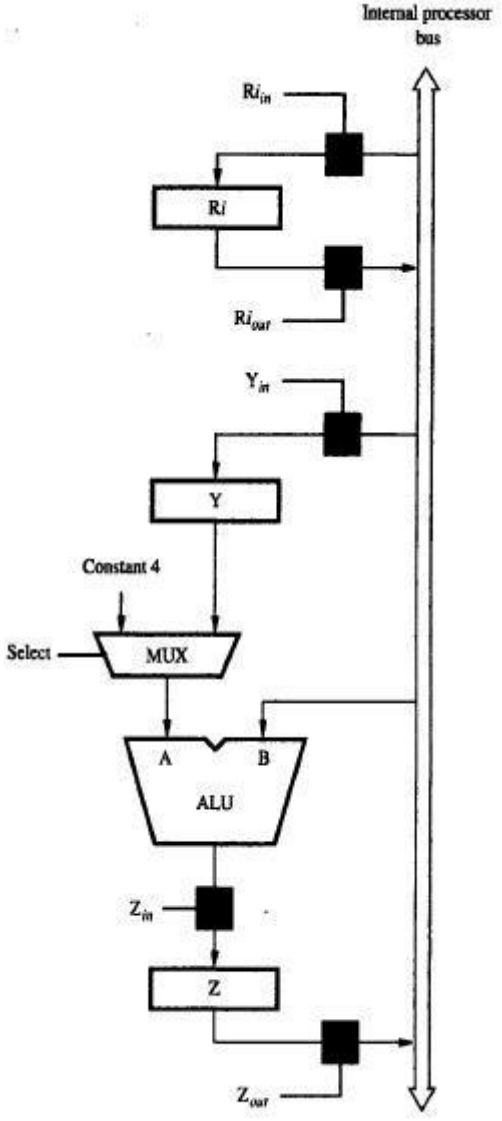


Figure 7.2 Input and output gating for the registers in Figure 7.1.

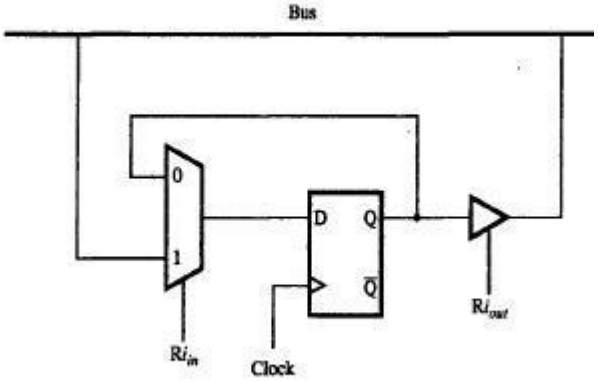


Figure 7.3 Input and output gating for one register bit.

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 &
  the other operand is obtained directly from 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 //transfer the contents of R1 to Y register
  2) R2out, SelectY, Add, Zin //R2 contents are transferred directly to B input of ALU.
     // The numbers of added. Sum stored in register Z
  3) Zout, R3in //sum is transferred to register R3
- The signals are activated for the duration of the clock cycle corresponding to that step. All other signals are inactive.

Write the complete control sequence for the instruction : Move (Rs),Rd

- This instruction copies the contents of memory-location pointed to by Rs into Rd. This is a memory read operation. This requires the following actions
  → fetch the instruction
  → fetch the operand (i.e. the contents of the memory-location pointed by Rs).
  → transfer the data to Rd.
- The control-sequence is written as follows
  1) PCout, MARin, Read, Select4, Add, Zin
  2) Z_{out}, PC_{in}, Y_{in}, WMFC
  3) MDR_{out}, IR_{in}
  4) R_s, MAR_{in}, Read
  5) MDRinE, WMFC
  6) MDRout, Rd, End

FETCHING A WORD FROM MEMORY

Q.2 Explain how read/fetch operation from memory takes place (or) Give control sequence for Move (R1),R2

• To fetch instruction/data from memory, processor transfers required address to MAR (whose output is connected to address-lines of memory-bus).

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

• MFC (Memory Function Completed): Addressed-device sets MFC to 1 to indicate that the contents of the specified location

→ have been read &

→ are available on data-lines of memory-bus

• Consider the instruction **Move (R1),R2.** The sequence of steps is:

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

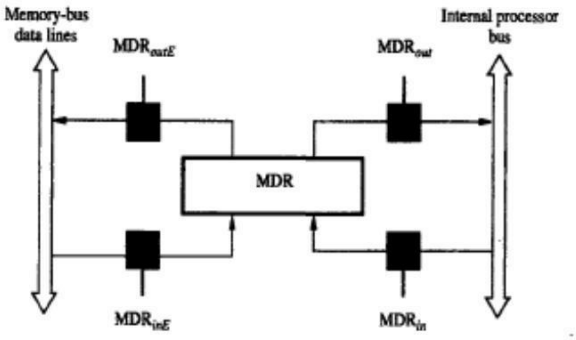


Figure 7.4 Connection and control signals for register MDR.

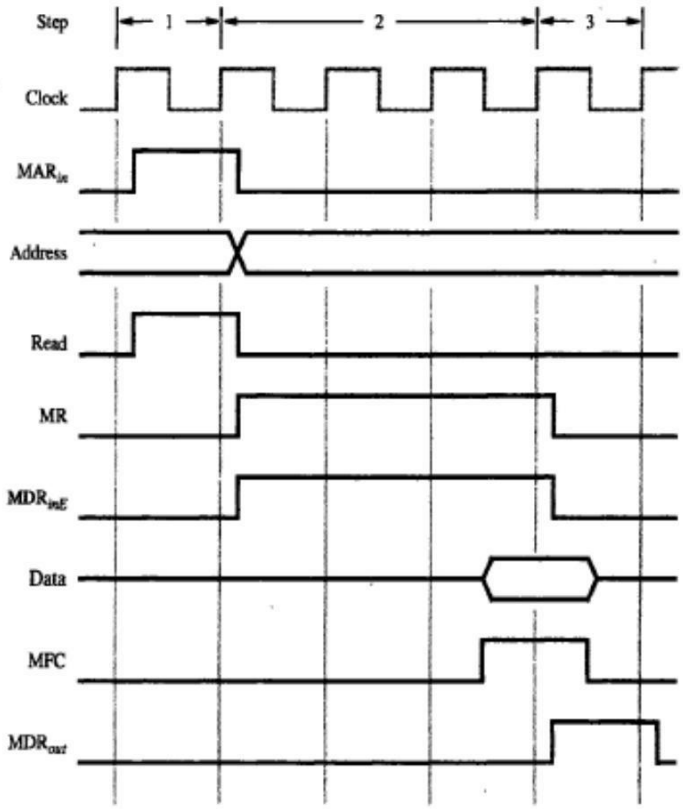


Figure 7.5 Timing of a memory Read operation.

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:

Q.3 What are actions required to execute a complete instruction Add(R3),R4

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

• Control sequence for execution of this instruction is as follows

Q. Give the control sequence for execution of instruction Add(R3),R4

1) PCout, MARin, Read, Select4, Add, Zin
2) Z_{out}, PC_{in}, Y_{in}, WMFC
3) MDRout, IRin
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

• 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.
Step3--> Fetched instruction is moved into MDR and then to IR.
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.

Write the complete control sequence for the instruction : Move (Rs),Rd

• This instruction copies the contents of memory-location pointed to by Rs into Rd. This is a memory read operation. This requires the following actions

→ fetch the instruction
→ fetch the operand (i.e. the contents of the memory-location pointed by Rs).
→ transfer the data to Rd.

• The control-sequence is written as follows

1) PCout, MARin, Read, Select4, Add, Zin
2) Z_{out}, PC_{in}, Y_{in}, WMFC
3) MDR_{out}, IR_{in}
4) R_s, MAR_{in}, Read
5) MDRinE, WMFC
6) MDRout, Rd, End

BRANCHING INSTRUCTIONS

• Control sequence for an unconditional branch instruction is as follows:

1) PCout, MARin, Read, Select4, Add, Zin
2) Z_{out}, PC_{in}, Y_{in}, WMFC

3) MDRout, IRin
4) Offset-field-of-IR_{out}, Add, Z_{in}
5) Z_{out}, PC_{in}, End

• The processing starts, as usual, the fetch phase ends in step3.
• In step 4, the offset-value is extracted from IR by instruction-decoding circuit.
• Since the updated value of PC is already available in register Y, the offset X is gated onto the bus, and an addition operation is performed.
• In step 5, the result, which is the branch-address, is loaded into the PC.
• The offset X used in a branch instruction is usually the difference between the branch target-address and the address immediately following the branch instruction. (For example, if the branch instruction is at location 1000 and branch target-address is 1200, then the value of X must be 196, since the PC will be containing the address 1004 after fetching the instruction at location 1000).
• In case of conditional branch, we need to check the status of the condition-codes before loading a new value into the PC.

e.g.: Offset-field-of-IRout, Add, Zin, If N=0 then End
If N=0, processor returns to step 1 immediately after step 4.
If N=1, step 5 is performed to load a new value into PC.

MULTIPLE BUS ORGANIZATION

Q.4 Draw and explain multiple bus organization of CPU. And write control sequence for the instruction Add R4,R5,R6.

• All general-purpose registers are combined into a single block called the *register file*.

• Register-file has 3 ports. There are 2 outputs allowing the contents of 2 different registers to be simultaneously placed on the buses A and B.

• Register-file has 3 ports.

1) Two output-ports allow the contents of 2 different registers to be simultaneously placed on buses A & B.
2) Third input-port allows data on bus C to be loaded into a third register during the same clock-cycle.

• Buses A and B are used to transfer source-operands to A & B inputs of ALU.

• Result is transferred to destination over bus C.

• Incrementer-unit is used to increment PC by 4.

• Control sequence for the instruction *Add R4,R5,R6* is as follows

1) PCout, R=B, MARin, Read, IncPC
2) WMFC
3) MDRout, R=B, IRin
4) $R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End

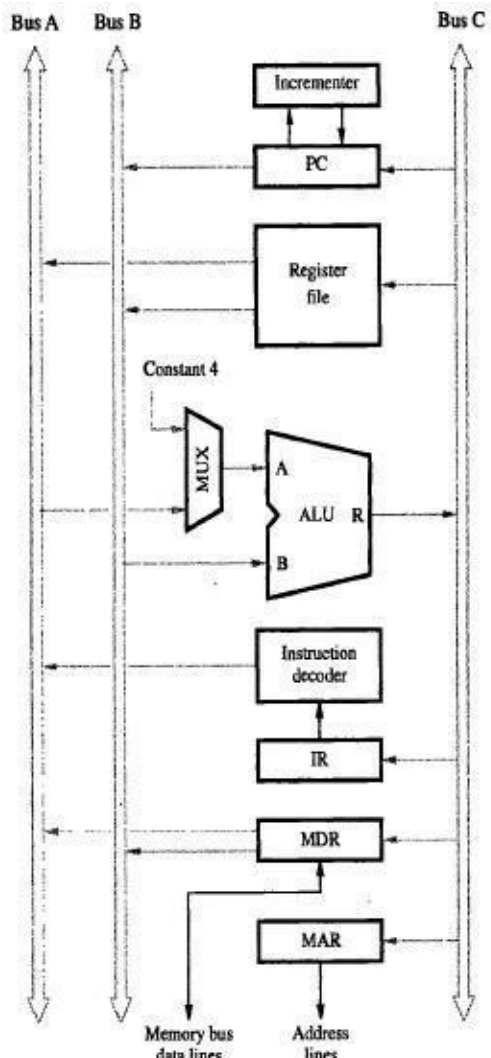
• Instruction execution proceeds as follows:

Step 1--> Contents of PC are passed through ALU using R=B control-signal and loaded into MAR to start a memory Read operation. At the same time, PC is incremented by 4.

Step2--> Processor waits for MFC signal from memory.

Step3--> Processor loads requested-data into MDR, and then transfers them to IR.

Step4--> The instruction is decoded and add operation take place in a single step.



Note:
To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence. There are two approaches for this purpose:

1) Hardwired control and 2) Microprogrammed control.

HARDWIRED CONTROL

Q.5 Explain Hardwired approach for generating control signals

• Decoder/encoder block is a combinational-circuit that generates required control-outputs depending on state of all its inputs.

• Step-decoder provides a separate signal line for each step in the control sequence.
Similarly, output of instruction-decoder consists of a separate line for each machine instruction.

• For any instruction loaded in IR, one of the output-lines INS1 through INSm is set to 1, and all other lines are set to 0.

• The input signals to encoder-block are combined to generate the individual control-signals Yin, PCout, Add, End and so on.

• For example, *Zin=T1+T6.ADD+T4.BR* ;This signal is asserted during time-slot T1 for all instructions,
during T6 for an Add instruction
during T4 for unconditional branch instruction

• When RUN=1, counter is incremented by 1 at the end of every clock cycle. When RUN=0, counter stops counting.

• Sequence of operations carried out by this machine is determined by wiring of logic elements, hence the name "hardwired".

• Advantage: Can operate at high speed.
Disadvantage: Limited flexibility.

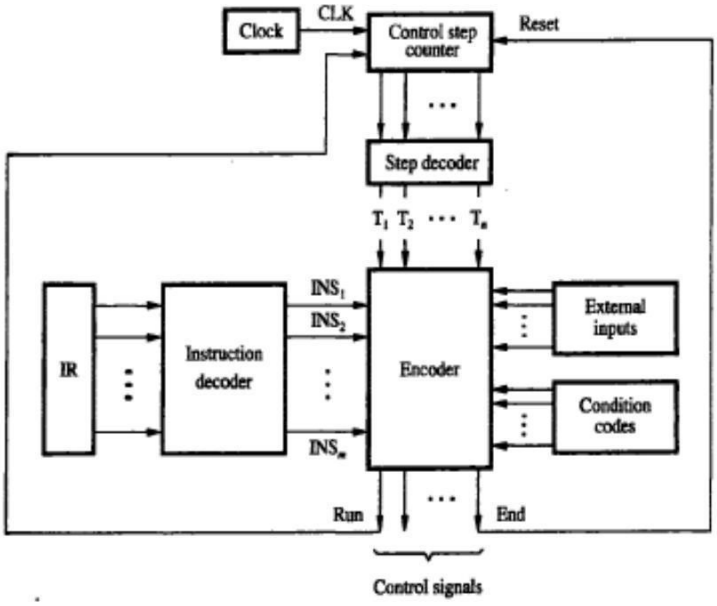


Figure 7.11 Separation of the decoding and encoding functions.

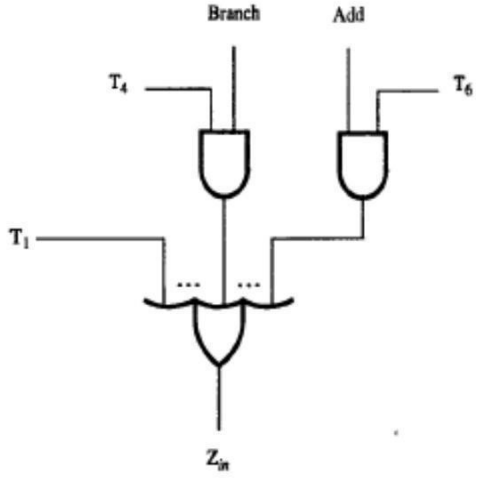


Figure 7.12 Generation of the Z_{in} control signal for the processor in Figure 7.1.

COMPLETE PROCESSOR

- This has separate processing-units to deal with integer data and floating-point data.
- A data-cache is inserted between these processing-units & main-memory.
- Instruction-unit fetches instructions
  → from an instruction-cache or
  → from main-memory when desired instructions are not already in cache
- Processor is connected to system-bus &
  hence to the rest of the computer by means of a bus interface
- Using separate caches for instructions & data is common practice in many processors today.
- A processor may include several units of each type to increase the potential for concurrent operations.

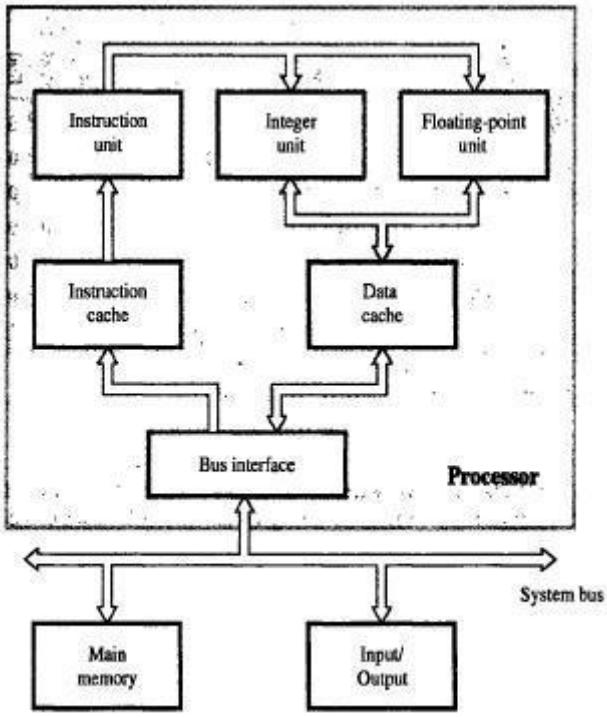


Figure 7.14 Block diagram of a complete processor.

MICROPROGRAMMED CONTROL

Q.6 Explain with a neat diagram micro programmed control method for design of control unit and write the micro routine for the instruction Branch<0

• Control-signals are generated by a program similar to machine language programs.
• *Control word(CW)* is a word whose individual bits represent various control-signals(like Add, End, Zin). {Each of the control-steps in control sequence of an instruction defines a unique combination of 1s & 0s in the CW}.
• Individual control-words in microroutine are referred to as *microinstructions.*
• A sequence of CWs corresponding to control-sequence of a machine instruction constitutes the *microroutine*.
• The microroutines for all instructions in the instruction-set of a computer are stored in a special memory called the *control store(CS).*
• Control-unit generates control-signals for any instruction by sequentially reading CWs of corresponding microroutine from CS.
• *Microprogram counter(μPC)* is used to read CWs sequentially from CS.
• Every time a new instruction is loaded into IR, output of "starting address generator" is loaded into μPC.
• Then, μPC is automatically incremented by clock,
causing successive microinstructions to be read from CS.
Hence, control-signals are delivered to various parts of processor in correct sequence.

Micro-instruction	..	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R3_{out}$	WMFC	End	..
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
4		0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	
5		0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	
6		0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	
7		0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	

Figure 7.15 An example of microinstructions for Figure 7.6.

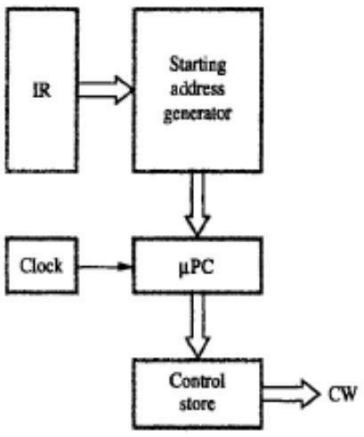


Figure 7.16 Basic organization of a microprogrammed control unit.

ORGANIZATION OF MICROPROGRAMMED CONTROL UNIT (TO SUPPORT CONDITIONAL BRANCHING)

• In case of conditional branching, microinstructions specify which of the external inputs, condition-codes should be checked as a condition for branching to take place.

• The *starting and branch address generator block* loads a new address into μPC when a microinstruction instructs it to do so.

• To allow implementation of a conditional branch, inputs to this block consist of

→ external inputs and condition-codes

→ contents of IR

• μPC is incremented every time a new microinstruction is fetched from microprogram memory except in following situations

i) When a new instruction is loaded into IR, μPC is loaded with starting-address of microroutine for that instruction.

ii) When a Branch microinstruction is encountered and branch condition is satisfied, μPC is loaded with branch-address.

iii) When an End microinstruction is encountered, μPC is loaded with address of first CW in microroutine for instruction fetch cycle.

Address	Microinstruction
0	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
1	Z_{out}, PC_{in}, Y_{in}, WMFC
2	MDR_{out}, IR_{in}
3	Branch to starting address of appropriate microroutine
...	...
25	If N=0, then branch to microinstruction 0
26	Offset-field-of-IR_{out}, SelectY, Add, Z_{in}
27	Z_{out}, PC_{in}, End

Figure 7.17 Microroutine for the instruction Branch < 0.

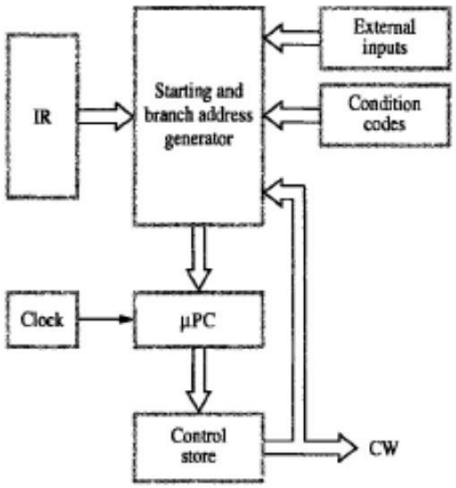


Figure 7.18 Organization of the control unit to allow conditional branching in the microprogram.

MICROINSTRUCTIONS

- Drawbacks of microprogrammed control:
    1) Assigning individual bits to each control-signal results in long microinstructions because the number of required signals is usually large.
    2) Available bit-space is poorly used because only a few bits are set to 1 in any given microinstruction.
- Solution: Signals can be grouped because
    1) Most signals are not needed simultaneously.
    2) Many signals are mutually exclusive.
- Grouping control-signals into fields requires a little more hardware because decoding-circuits must be used to decode bit patterns of each field into individual control signals.
- Advantage: This method results in a smaller control-store (only 20 bits are needed to store the patterns for the 42 signals).

Vertical organization	Horizontal organization
Highly encoded schemes that use compact codes to specify only a small number of control functions in each microinstruction are referred to as a vertical organization	The minimally encoded scheme in which many resources can be controlled with a single microinstuction is called a horizontal organization
This approach results in considerably slower operating speeds because more micrinstructions are needed to perform the desired control functions	This approach is useful when a higher operating speed is desired and when the machine structure allows parallel use of resources

Microinstruction

F1	F2	F3	F4	F5
F1 (4 bits)	F2 (3 bits)	F3 (3 bits)	F4 (4 bits)	F5 (2 bits)
0000: No transfer 0001: PC_{out} 0010: MDR_{out} 0011: Z_{out} 0100: $R0_{out}$ 0101: $R1_{out}$ 0110: $R2_{out}$ 0111: $R3_{out}$ 1010: $TEMP_{out}$ 1011: $Offset_{out}$	000: No transfer 001: PC_{in} 010: IR_{in} 011: Z_{in} 100: $R0_{in}$ 101: $R1_{in}$ 110: $R2_{in}$ 111: $R3_{in}$	000: No transfer 001: MAR_{in} 010: MDR_{in} 011: $TEMP_{in}$ 100: Y_{in}	0000: Add 0001: Sub ⋮ 1111: XOR 16 ALU functions	00: No action 01: Read 10: Write

F6	F7	F8	...
F6 (1 bit)	F7 (1 bit)	F8 (1 bit)	
0: SelectY 1: Select4	0: No action 1: WMFC	0: Continue 1: End	

Figure 7.19 An example of a partial format for field-encoded microinstructions.

MICROPROGRAM SEQUENCING

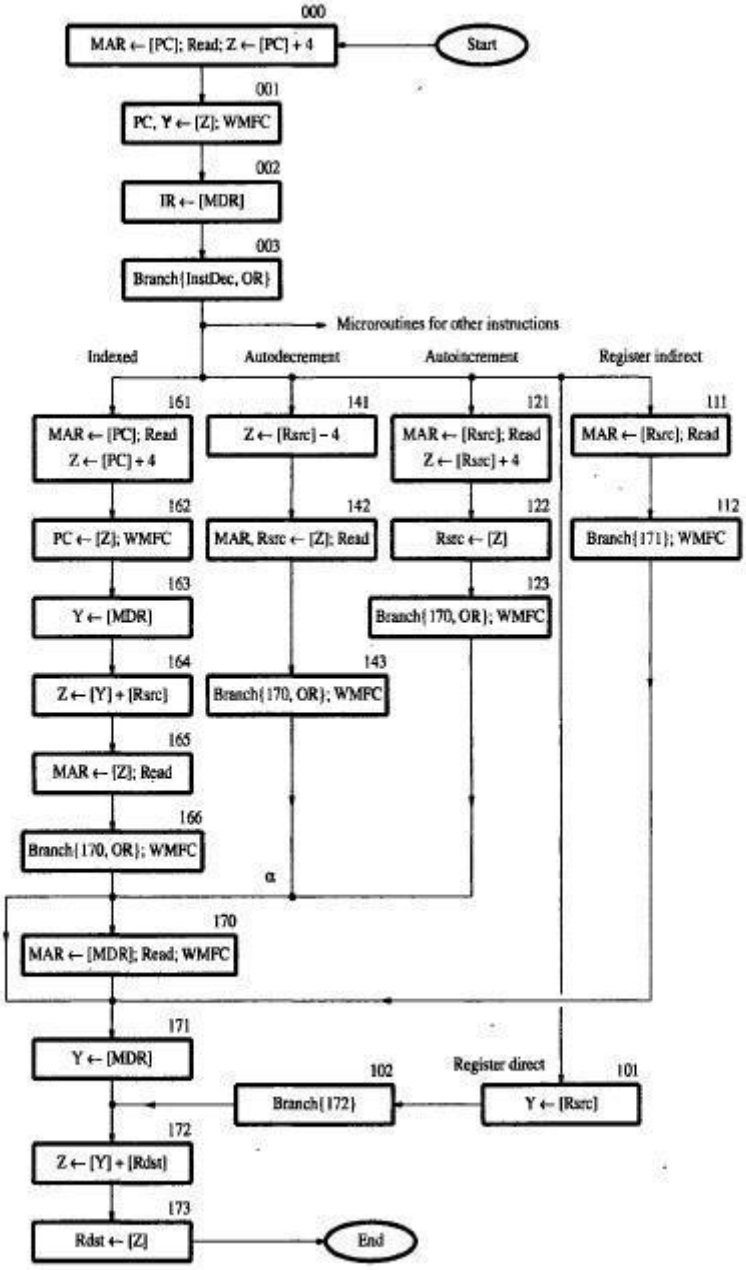


Figure 7.20 Flowchart of a microprogram for the Add src,Rdst instruction.

• Two major disadvantage of microprogrammed control is:

1) Having a separate microroutine for each machine instruction results in a large total number of microinstructions and a large control-store.

2) Execution time is longer because it takes more time to carry out the required branches.

• Consider the instruction *Add src,Rdst* ;which adds the source-operand to the contents of Rdst and places the sum in Rdst.

• Let source-operand can be specified in following addressing modes: register, autoincrement, autodecrement and indexed as well as the indirect forms of these 4 modes.

• Each box in the chart corresponds to a microinstruction that controls the transfers and operations indicated within the box.

• The microinstruction is located at the address indicated by the octal number (001,002).

BRANCH ADDRESS MODIFICATION USING BIT-ORING

• Consider the point labeled α in the figure. At this point, it is necessary to choose between direct and indirect addressing modes.
• If indirect-mode is specified in the instruction, then the microinstruction in location 170 is performed to fetch the operand from the memory.
If direct-mode is specified, this fetch must be bypassed by branching immediately to location 171.
• The most efficient way to bypass microinstruction 170 is to have the preceding branch microinstructions specify the address 170 and then use an OR gate to change the LSB of this address to 1 if the direct addressing mode is involved. This is known as the *bit-ORing* technique.

WIDE BRANCH ADDRESSING

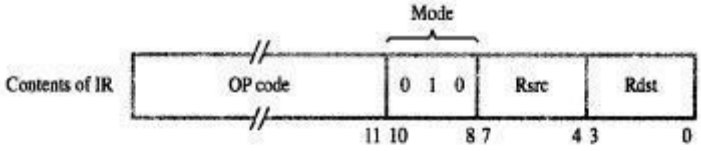
• The instruction-decoder(InstDec) generates the starting-address of the microroutine that implements the instruction that has just been loaded into the IR.
• Here, register IR contains the Add instruction, for which the instruction decoder generates the microinstruction address 101. (However, this address cannot be loaded as is into the μPC).
• The source-operand can be specified in any of several addressing-modes. The bit-ORing technique can be used to modify the starting-address generated by the instruction-decoder to reach the appropriate path.

Use of WMFC

• WMFC signal is issued at location 112 which causes a branch to the microinstruction in location 171.
• WMFC signal means that the microinstruction may take several clock cycles to complete. If the branch is allowed to happen in the first clock cycle, the microinstruction at location 171 would be fetched and executed prematurely.
To avoid this problem, WMFC signal must inhibit any change in the contents of the μPC during the waiting-period.

Detailed Examination

• Consider *Add (Rsrc)+,Rdst;* which adds Rsrc content to Rdst content, then stores the sum in Rdst and finally increments Rsrc by 4 (i.e. auto-increment mode).
• In bit 10 and 9, bit-patterns 11, 10, 01 and 00 denote indexed, auto-decrement, auto-increment and register modes respectively. For each of these modes, bit 8 is used to specify the indirect version.
• The processor has 16 registers that can be used for addressing purposes; each specified using a 4-bit-code.
• There are 2 stages of decoding:
1) The microinstruction field must be decoded to determine that an Rsrc or Rdst register is involved.
2) The decoded output is then used to gate the contents of the Rsrc or Rdst fields in the IR into a second decoder, which produces the gating-signals for the actual registers R0 to R15.



Address (octal)	Microinstruction
000	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
001	Z_{out}, PC_{in}, Y_{in}, WMFC
002	MDR_{out}, IR_{in}
003	μBranch {μPC ← 101 (from Instruction decoder); $\mu PC_{5,4} \leftarrow [IR_{10,9}]$; $\mu PC_3 \leftarrow [\overline{IR_{10}}] \cdot [\overline{IR_9}] \cdot [IR_8]$}
121	$Rsrc_{out}$, MAR_{in}, Read, Select4, Add, Z_{in}
122	Z_{out}, $Rsrc_{in}$
123	μBranch {μPC ← 170; $\mu PC_0 \leftarrow [\overline{IR_8}]$}, WMFC
170	MDR_{out}, MAR_{in}, Read, WMFC
171	MDR_{out}, Y_{in}
172	$Rdst_{out}$, SelectY, Add, Z_{in}
173	Z_{out}, $Rdst_{in}$, End

MICROINSTRUCTIONS WITH NEXT-ADDRESS FIELDS

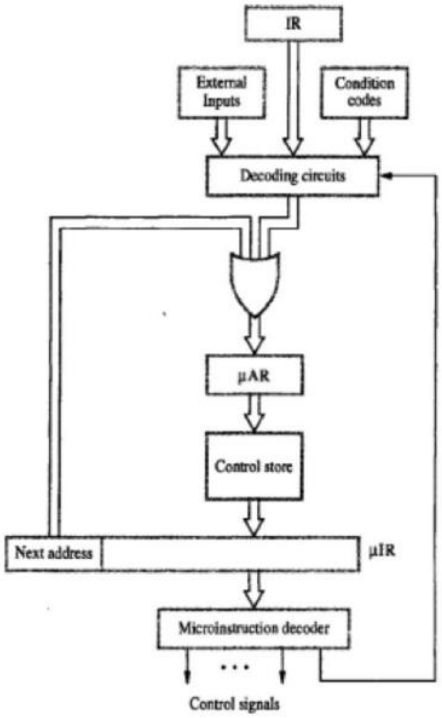


Figure 7.22 Microinstruction-sequencing organization.

Microinstruction

F0	F1	F2	F3
F0 (8 bits)	F1 (3 bits)	F2 (3 bits)	F3 (3 bits)
Address of next microinstruction	000: No transfer 001: PC_{out} 010: MDR_{out} 011: Z_{out} 100: $Rsrc_{out}$ 101: $Rdst_{out}$ 110: $TEMP_{out}$	000: No transfer 001: PC_{in} 010: IR_{in} 011: Z_{in} 100: $Rsrc_{in}$ 101: $Rdst_{in}$	000: No transfer 001: MAR_{in} 010: MDR_{in} 011: $TEMP_{in}$ 100: Y_{in}

F4	F5	F6	F7
F4 (4 bits)	F5 (2 bits)	F6 (1 bit)	F7 (1 bit)
0000: Add 0001: Sub ⋮ 1111: XOR	00: No action 01: Read 10: Write	0: SelectY 1: Select4	0: No action 1: WMFC

F8	F9	F10
F8 (1 bit)	F9 (1 bit)	F10 (1 bit)
0: NextAdrs 1: InstDec	0: No action 1: OR_{mode}	0: No action 1: OR_{indsrc}

Figure 7.23 Format for microinstructions in the example of Section 7.5.3.

Octal address	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
000	00000001	001	011	001	0000	01	1	0	0	0	0
001	00000010	011	001	100	0000	00	0	1	0	0	0
002	00000011	010	010	000	0000	00	0	0	0	0	0
003	00000000	000	000	000	0000	00	0	0	1	1	0
121	01010010	100	011	001	0000	01	1	0	0	0	0
122	01111000	011	100	000	0000	00	0	1	0	0	1
170	01111001	010	000	001	0000	01	0	1	0	0	0
171	01111010	010	000	100	0000	00	0	0	0	0	0
172	01111011	101	011	000	0000	00	0	0	0	0	0
173	00000000	011	101	000	0000	00	0	0	0	0	0

Figure 7.24 Implementation of the microroutine of Figure 7.21 using a next-microinstruction address field. (See Figure 7.23 for encoded signals.)

• The microprogram requires several branch microinstructions which perform no useful operation. Thus, they detract from the operating speed of the computer.
• Solution: Include an address-field as a part of every microinstruction to indicate the location of the next microinstruction to be fetched. (This means every microinstruction becomes a branch microinstruction).
• The flexibility of this approach comes at the expense of additional bits for the address-field.
• Advantage: Separate branch microinstructions are virtually eliminated. There are few limitations in assigning addresses to microinstructions. There is no need for a counter to keep track of sequential addresse. Hence, the
μPC is replaced with a μAR (Microinstruction Address Register). {which is loaded from the next-address field in each microinstruction}.
• The next-address bits are fed through the OR gate to the μAR, so that the address can be modified on the basis of the data in the IR, external inputs and condition-codes.
• The decoding circuits generate the starting-address of a given microroutine on the basis of the opcode in the IR.

PREFETCHING MICROINSTRUCTIONS

• Drawback of microprogrammed control: Slower operating speed because of the time it takes to fetch microinstructions from the control-store.

• Solution: Faster operation is achieved if the next microinstruction is pre-fetched while the current one is being executed.

Emulation

• The main function of microprogrammed control is to provide a means for simple, flexible and relatively inexpensive execution of machine instruction.

• Its flexibility in using a machine's resources allows diverse classes of instructions to be implemented.

• Suppose we add to the instruction-repository of a given computer M1, an entirely new set of instructions that is in fact the instruction-set of a different computer M2.

• Programs written in the machine language of M2 can be then be run on computer M1 i.e. M1 emulates M2.

• Emulation allows us to replace obsolete equipment with more up-to-date machines.

• If the replacement computer fully emulates the original one, then no software changes have to be made to run existing programs.

• Emulation is easiest when the machines involved have similar architectures.

Embedded Systems

- A physical system that employs computer control for a specific purpose, rather than for general-purpose computation, is referred to as an *embedded system*.
- Microprocessor control is now commonly used in cameras, cell phones, display phones, point-of-sale terminals, kitchen appliances, cars, and many toys. Low cost and high reliability are the essential requirements in these applications. Small size and low power consumption are often of key importance.
- All of this can be achieved by placing on a single chip not only the processor circuitry, but also some memory, input/output
- interfaces, timer circuits, and other features to make it easy to implement a complete computer control system using very few chips. Microprocessor chips of this type are generally referred to as *microcontrollers*

Examples of Embedded Systems

Microwave Oven

Q.7 Briefly explain the block diagram of Microwave oven

- Household appliance microwave oven uses computer control to govern their operation.
- This appliance is based on a **magnetron** power unit that generates the microwaves used to heat food in a confined space.
- Power levels are achieved by turning the magnetron on and off for controlled time intervals.
- By controlling the power level and the total heating time, it is possible to realize a variety of user-selectable cooking options.

Specification for a microwave

1. Manual selection of the power level and cooking time
2. Manual selection of the sequence of different cooking steps
3. Automatic operation, where the user specifies the type of food and the weight of the food; then an appropriate power level and time are calculated by the controller
4. Automatic defrosting of food by specifying the weight

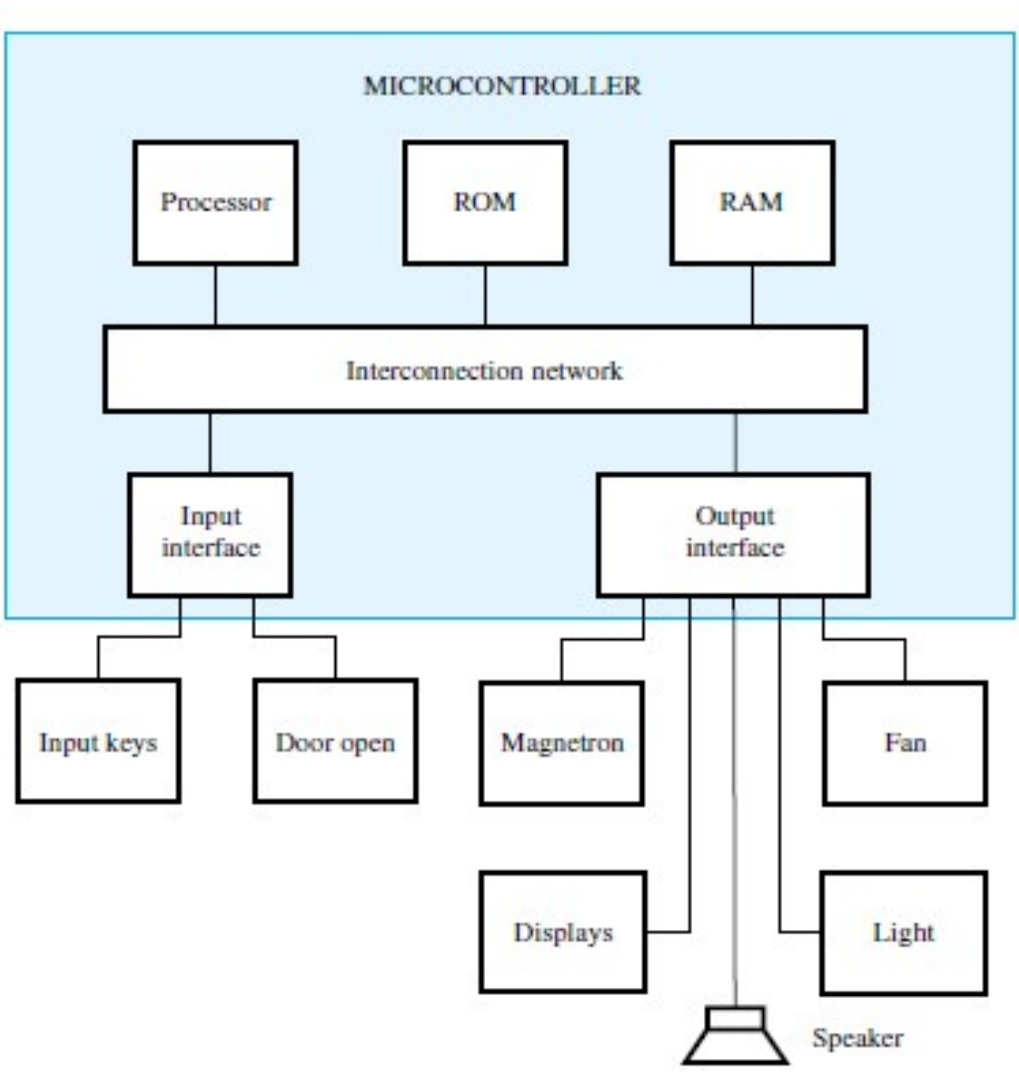


Figure 10.1 A block diagram of a microwave oven.

The oven includes a display that can show:

- Time-of-day clock
- Decrementing clock timer while cooking
- Information messages to the user

- An audio alert signal, in the form of a beep tone, is used to indicate the end of a cooking operation.
- An exhaust fan and oven light are provided. As a safety measure, a door interlock turns the magnetron off if the door of the oven is open. All of these functions can be controlled by a microcontroller.

The input/output capability needed to communicate with the user includes:

- Input keys that comprise a 0 to 9 number pad.
- Function keys such as Reset, Start, Stop, Power Level, Auto Defrost, Auto Cooking, Clock Set, and Fan Control
- Visual output in the form of a liquid-crystal display, similar to the seven-segment display.
- A small speaker that produces the beep tone

The computational tasks executed by a microcontroller to control a microwave oven

- Maintaining the time-of-day clock,
- Determining the actions needed for the various cooking options,
- Generating the control signals needed to turn on or off devices such as the magnetron and the fan

generating display information.

- The program needed to implement the desired actions is quite small. It is stored in a non-volatile read-only
- Small RAM is used during computations to hold the user-entered data.

Digital Camera

Q.8 Briefly explain the block diagram of Digital Camera

- Traditional cameras use film to capture images.
- In a digital camera, an array of optical sensors is used to capture images. These sensors convert light into electrical charge.
- The intensity of light determines the amount of charge that is generated.
- Two different types of sensors are used in commercial products.
    1. *charge-coupled devices* (CCDs).
    2. Sensors based on CMOS technology

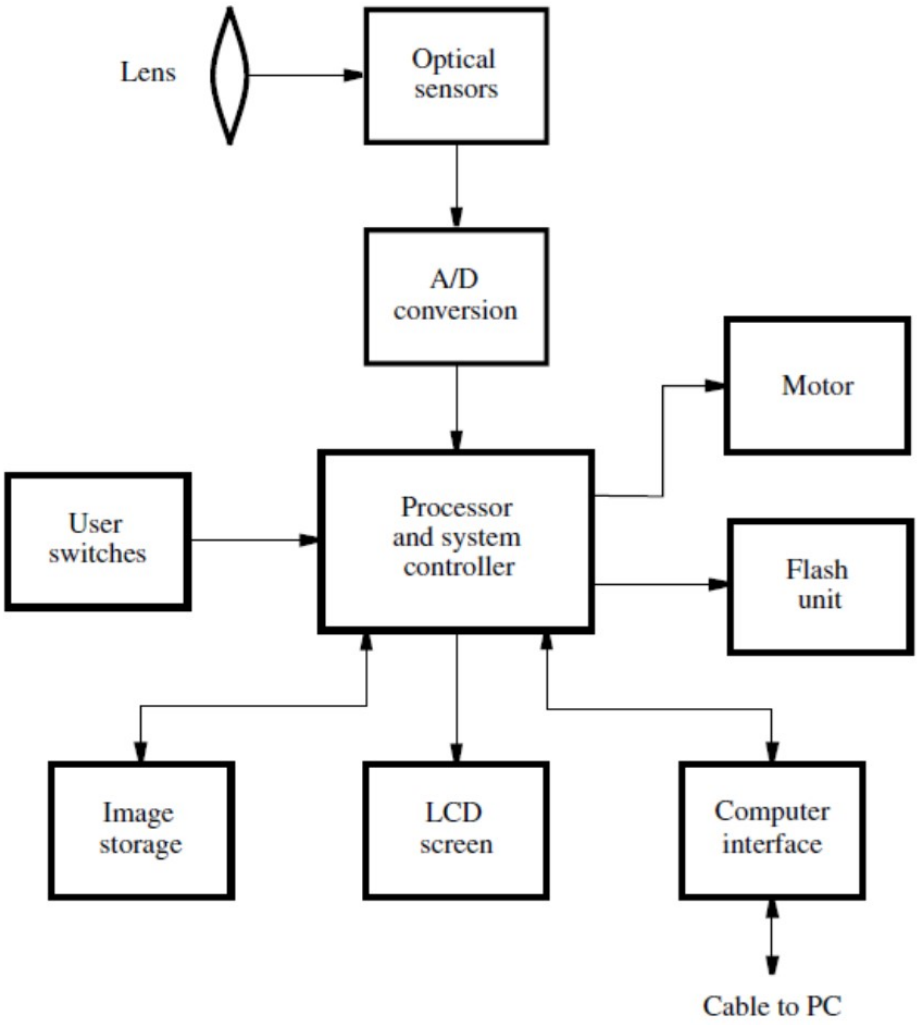


Figure 10.2 A simplified block diagram of a digital camera.

- Each sensing element generates a charge that corresponds to one *pixel,* which is one point of a pictorial image.
- The number of pixels determines the quality of pictures.

- The charge is an analog quantity, which is converted into a digital representation using *analog-to-digital* (A/D) conversion circuits.
- A/D conversion produces a digital representation of the image in which the color and intensity of each pixel are represented by a number of bits.
- The processor and system controller block includes a variety of interface circuits needed to connect to other parts of the system.
- The processor governs the operation of the camera.
- It processes the raw image data obtained from the A/D circuits to generate images represented in standard formats suitable for use in computers, printers, and display devices.
- The main formats used are TIFF (Tagged Image File Format) for uncompressed images and JPEG (Joint Photographic Experts Group) for compressed images.
- The processed images are stored in a larger image storage device. Flash memory cards, are a popular choice for storing images.
- A captured and processed image can be displayed on a liquid-crystal display (LCD) screen, which is included in the camera.
- The number of images that can be saved depends on the size of the image storage unit.
- It also depends on the chosen quality of the images, namely on the number of pixels per image and on the degree of compression (for JPEG format).
- A standard interface provides a mechanism for transferring the images to a computer or a printer using a USB cable.
- The system controller generates the signals needed to control the operation of the focusing mechanism and the flash unit.
- A digital camera requires a considerably more powerful processor as it has to perform complex signal processing functions.
- Typically, the processor consumes less power than the display and flash units of a camera.

Home Telemetry

- Using signals from a remote location to observe and control the state of equipment is often referred to as ***telemetry***.
- Home example equipment: washers, dryers, dishwashers, cooking ranges, furnaces, and air conditioners.
- Another notable example is the display telephone.
- In addition to the **standard telephone features, a telephone with an embedded microcontroller** can be used to provide remote access to other devices in the home.
- **Using the telephone one can remotely perform functions such as:**
  1. Communicate with a computer-controlled home security system
  2. Set a desired temperature to be maintained by a furnace or an air conditioner
  3. Set the start time, the cooking time, and the temperature for food that has been placed in the oven at some earlier time
  4. Read the electricity, gas, and water meters, replacing the need for the utility companies to send an employee to the home to read the meters

Microcontroller Chips for Embedded Applications

- A microcontroller chip should be versatile enough to serve a wide variety of applications.
- The main part is a processor core, which may be a basic version of a commercially available microprocessor.
- RAM Memory to hold the data that change during computations.
- Read-only(EEPROM and Flash memory) to hold the software,

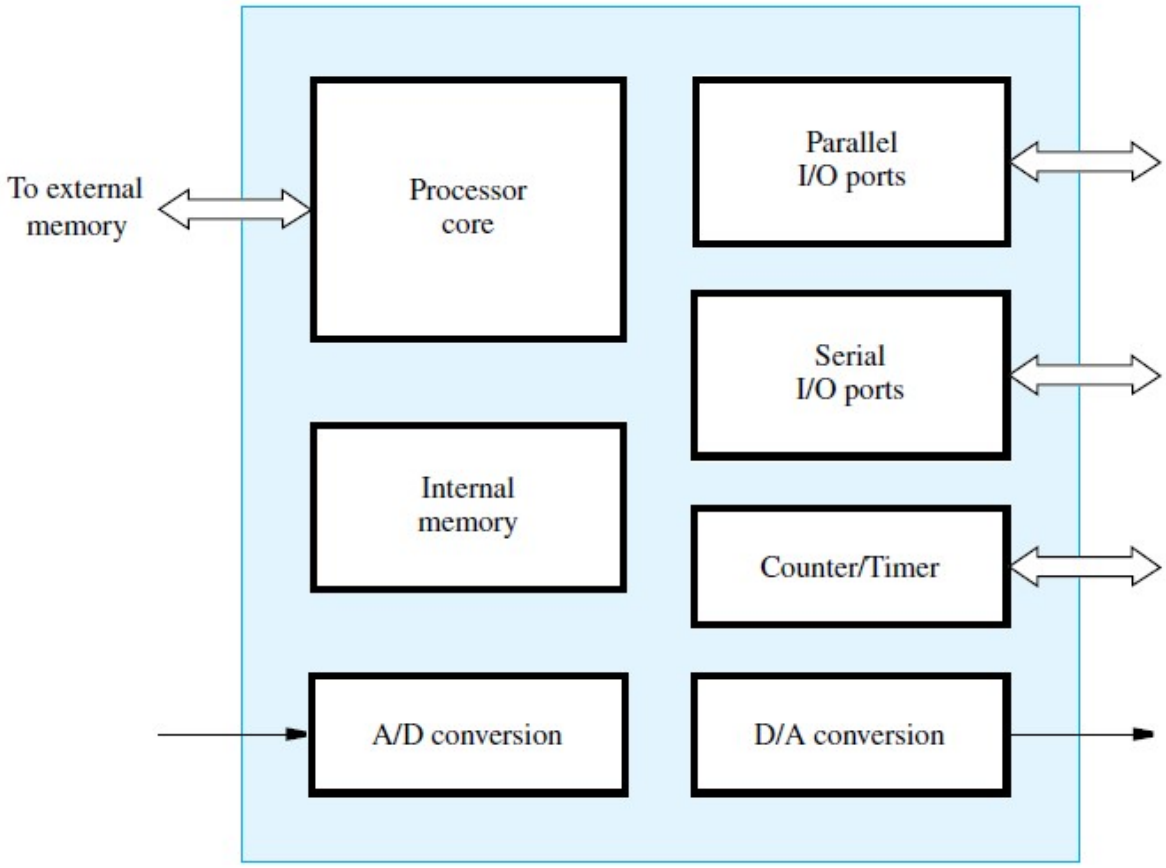


Figure 10.3 A block diagram of a microcontroller.

- Several **I/O ports** are usually provided for both parallel and serial interfaces.
- In many applications, it is necessary to generate control signals at programmable time intervals. This task is achieved easily if a **timer circuit** is included in the microcontroller chip.
- An embedded system may include some analog devices. To deal with such devices, it is necessary to be able to convert analog signals into digital representations, and vice versa. This is conveniently accomplished if the embedded controller includes **A/D and D/A conversion circuits**.
- Many embedded processor chips are available commercially.
- Examples are: Freescale's 68HC11 and 68K/ColdFire families, Intel's 8051 and MCS-96 families,

A Simple Microcontroller

- There is a **processor core and some on-chip memory** that may not be sufficient to support all potential applications, so **External memory** can be added.
- There are two 8-bit parallel interfaces, called port A and port B, and one serial interface.
- The microcontroller also contains a 32-bit counter/timer circuit, which can be used to generate internal interrupts at programmed time intervals, to serve as a system timer, to count the pulses on an input line, to generate square-wave output signals, and so on.

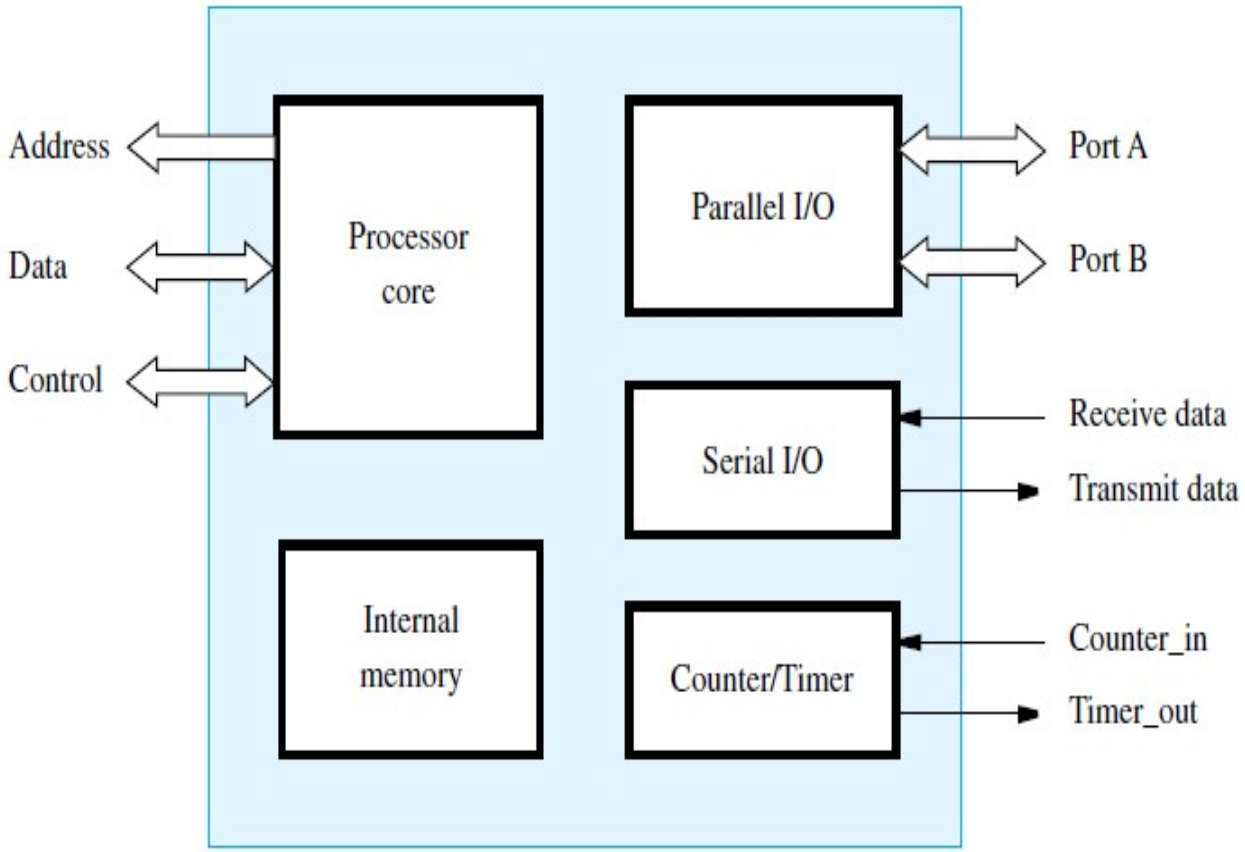


Figure 10.4 An example microcontroller.

Parallel I/O Interface

Q.9 With a block diagram explain parallel I/O interface.

- Embedded system applications require considerable flexibility in input/output interfaces.
- One of the objectives of the design of input/output interfaces for a microcontroller is to reduce the need for external circuitry as much as possible.
- Each parallel port has an associated eight-bit data direction register, which can be used to configure individual data lines as either input or output.
- Port pin PA*i* is treated as an input if the data direction flip-flop contains a 0.
- In this case, activation of the control signal Read_Port places the logic value on the port pin onto the data line D*i* of the processor bus.
- The port pin serves as an output if the data direction flip-flop is set to 1. The value loaded into the output data flip-flop, under control of the Write_Port signal, is placed on the pin.

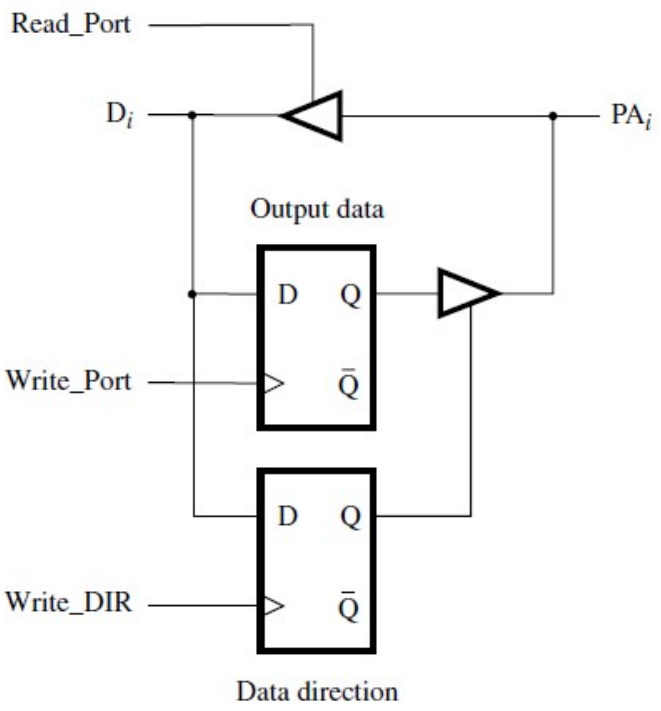


Figure 10.5 Access to one bit in port A in Figure 10.4.

Figure 10.6 depicts all registers in the parallel interface, as well as the addresses assigned to them. We have arbitrarily chosen addresses at the high end of a 32-bit address range. The information in the status and control registers is used for controlling data transfers to and from the devices connected to ports A and B. The status register, PSTAT, contains the status flags.

- The PASIN flag is set to 1 when there are new data on port A. It is cleared to 0 when the processor accepts the data by reading the PAIN register.
- The PASOUT flag is set to 1 when the data in register PAOUT are accepted by the connected device, to indicate that the processor may now load new data into PAOUT. The PASOUT flag is cleared to 0 when the processor writes data into PAOUT.
- The status register also contains four interrupt flags.
- An interrupt flag, such as IAIN, is set to 1 when that interrupt is enabled and the corresponding I/O action occurs. The interrupt-enable bits are held in control register PCONT. An enable bit is set to 1 to enable the corresponding interrupt. For example, if ENAIN=1 and PASIN=1, then the interrupt flag IAIN is set to 1 and an interrupt request is raised. Thus,
- IAIN = ENAIN · PASIN
- The flags PBSIN and PBSOUT perform the same function for port B. Control register bits PAREG and PBREG are used to select the mode of operation of inputs to ports A and B, respectively. If set to 1, a register is used to store the input data; otherwise, a direct path from the pins is used as indicated in Figure 10.5.

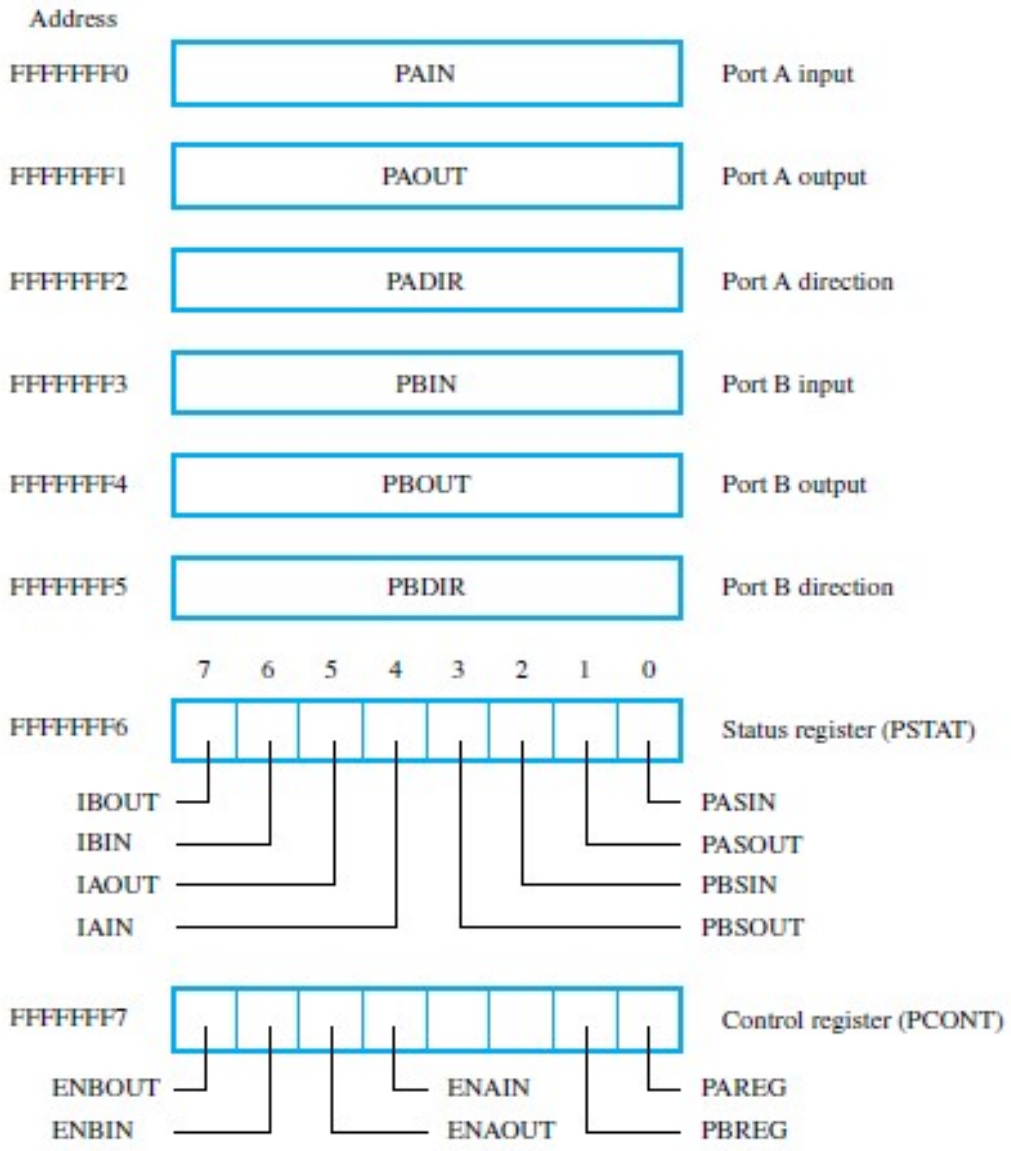


Figure 10.6 Parallel interface registers.

Serial I/O Interface

Q.10 With a block diagram explain serial I/O interface.

The serial interface provides the UART (Universal Asynchronous Receiver/Transmitter) capability to transfer data based on the scheme described in Section 7.4.2. Double buffering is used in both the transmit and receive paths, as shown in Figure 10.7. Such buffering is needed to handle bursts in I/O transfers correctly.

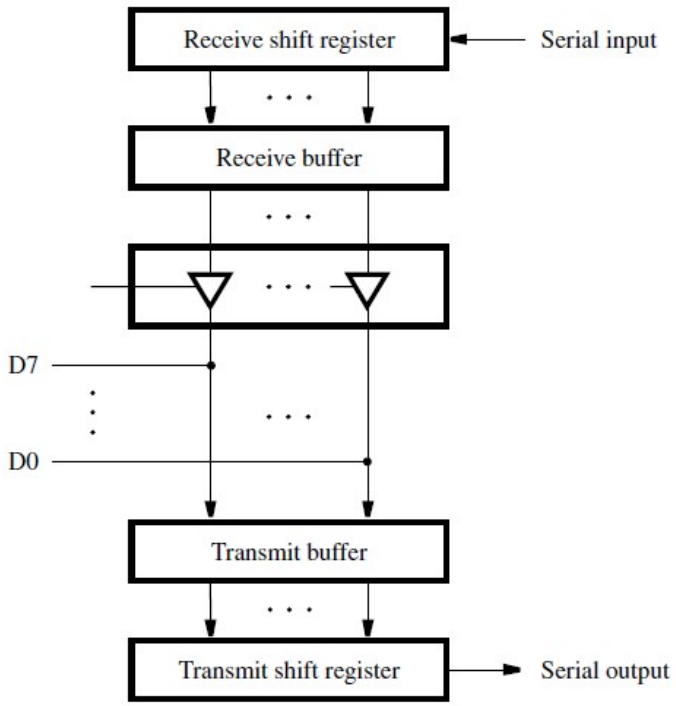


Figure 10.7 Receive and transmit structure of the serial interface.

- Input data are read from the 8-bit Receive buffer, and output data are loaded into the 8-bit Transmit buffer.
- The status register, SSTAT, provides information about the current status of the receive and transmit units.
- Bit SSTAT0 is set to 1 when there are valid data in the receive buffer; it is cleared to 0 automatically upon a read access to the receive buffer.
- Bit SSTAT1 is set to 1 when the transmit buffer is empty and can be loaded with new data.
- Bit SSTAT2 is set to 1 if an error occurs during the receive process.
- Status register also contains the interrupt flags.
- Bit SSTAT4 is set to 1 when the receive buffer becomes full and the receiver interrupt is enabled. Similarly, SSTAT5 is set to 1 when the transmit buffer becomes empty and the transmitter interrupt is enabled.
- The serial interface raises an interrupt if either SSTAT4 or SSTAT5 is equal to 1.
- It also raises an interrupt if SSTAT6 = 1, which occurs if SSTAT2 = 1 and the error condition interrupt is enabled.
- The control register, SCONT, is used to hold the interrupt-enable bits. Setting bits
- SCONT6−4 to 1 or 0 enables or disables the corresponding interrupts, respectively. This register also indicates how the transmit clock is generated.
- If SCONT0 = 0, then the transmit clock is the same as the system (processor) clock. If SCONT0 = 1, then a lower frequency transmit clock is obtained using a clock-dividing circuit.
- The last register in the serial interface is the clock-divisor register, DIV. This 32-bit register is associated with a counter circuit that divides down the system clock signal to generate the serial transmission clock.

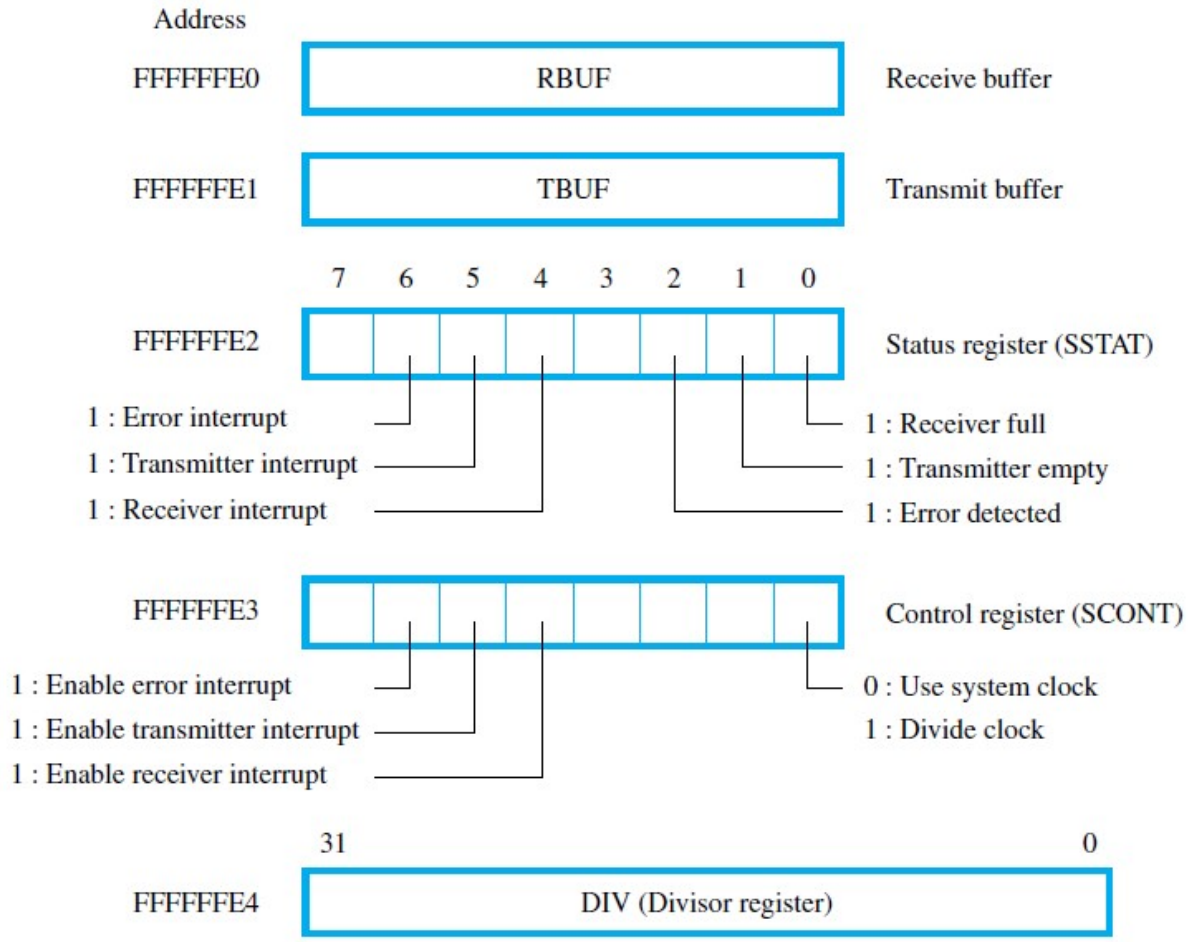


Figure 10.8 Serial interface registers.

Counter/Timer

Q.11 With a block diagram explain Counter/Timer interface.

- A 32-bit down-counter circuit is provided for use as either a counter or a timer.
- The basic operation of the circuit involves loading a starting value into the counter, and then decrementing the counter contents using either the internal system clock or an external clock signal.
- CNTM, can be loaded with an initial value, which is then transferred
- into the counter circuit.
- The current contents of the counter can be read by accessing memory address FFFFFFD4.
- The control register, CTCON, is used to specify the operating mode of the counter/timer circuit. It provides a mechanism for starting and stopping the counting process, and for enabling interrupts when the counter contents are decremented to 0.
- The status register, CTSTAT, reflects the state of the circuit.

Counter Mode

The counter mode is selected by setting bit $CTCON_7$ to 0. The starting value is loaded into the counter by writing it into register CNTM. The counting process begins when bit $CTCON_0$ is set to 1 by a program instruction. Once counting starts, bit $CTCON_0$ is automatically cleared to 0. The counter is decremented by pulses on the Counter_in line in Figure 10.4. Upon reaching 0, the counter circuit sets the status flag $CTSTAT_0$ to 1, and raises an interrupt if the corresponding interrupt-enable bit has been set to 1. The next clock pulse causes the counter to reload the starting value, which is held in register CNTM, and counting continues. The counting process is stopped by setting bit $CTCON_1$ to 1.

Timer Mode

The timer mode is selected by setting bit $CTCON_7$ to 1. This mode can be used to generate periodic interrupts. It is also suitable for generating a square-wave signal on the output line Timer_out in Figure 10.4. The process starts as explained above for the counter mode. As the counter counts down, the value on the output line is held constant. Upon reaching zero, the counter is reloaded automatically with the starting value, and the output.

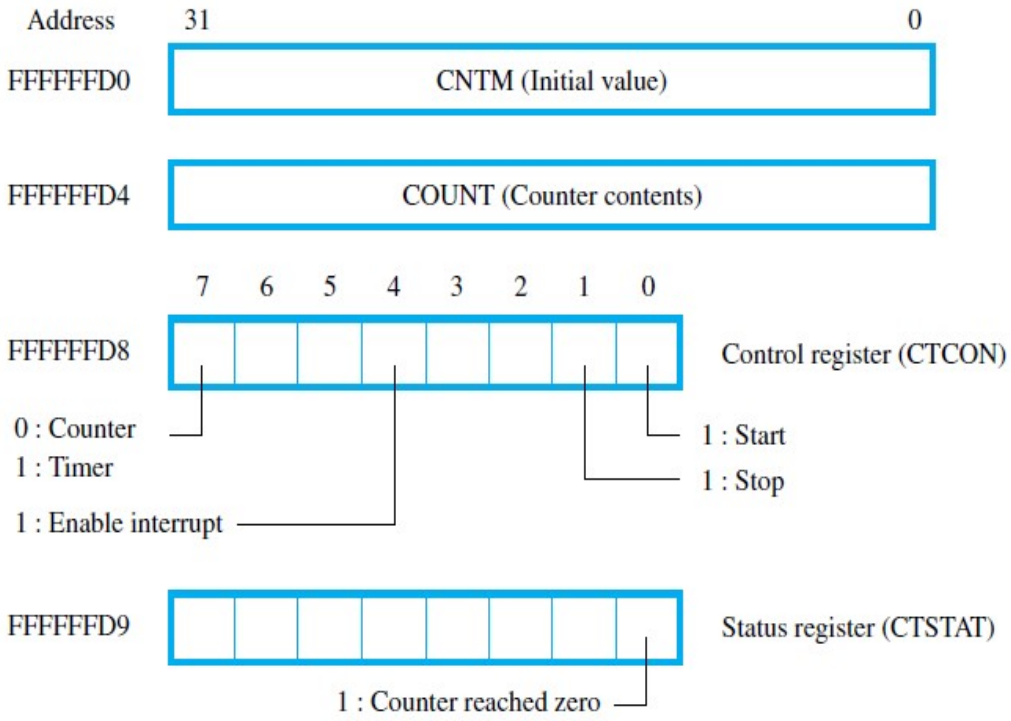


Figure 10.9 Counter/Timer registers.

Structure of General Purpose Multiprocessor

Q. 12Explain multiprocessor. Justify how delay is reduced.

- A multiprocessor system consists of a number of processors capable of simultaneously executing independent tasks.
- In a shared-memory multiprocessor, all processors have access to the same memory. Tasks running in different processors can access shared variables in the memory using the same addresses. The size of the shared memory is likely to be large. Implementing a large memory in a single module would create a bottleneck when many processors make requests to access the memory simultaneously.
- This problem is alleviated by distributing the memory across multiple modules so that simultaneous requests from different processors are more likely to access different memory modules, depending on the addresses of those requests.
- An interconnection network enables any processor to access any module that is a part of the shared memory. When memory modules are kept physically separate from the processors, all requests to access memory must pass through the network, which introduces latency. Figure 12.2 shows such an arrangement. A system which has the same network (delay) latency for all accesses from the processors to the memory modules is called a Uniform Memory Access (UMA) multiprocessor. Although the latency is uniform, it may be large for a network that connects many processors and memory modules.
- For better performance, it is desirable to place a memory module close to each processor. The result is a collection of *nodes*, each consisting of a processor and a memory module. The nodes are then connected to the network, as shown in Figure 12.3. The network latency (delay) is avoided when a processor makes a request to access its local memory. However, a request to access a remote memory module must pass through the network. Because of the difference in latencies for accessing local and remote portions of the shared memory, systems of this type are called Non-Uniform Memory Access (NUMA) multiprocessors.

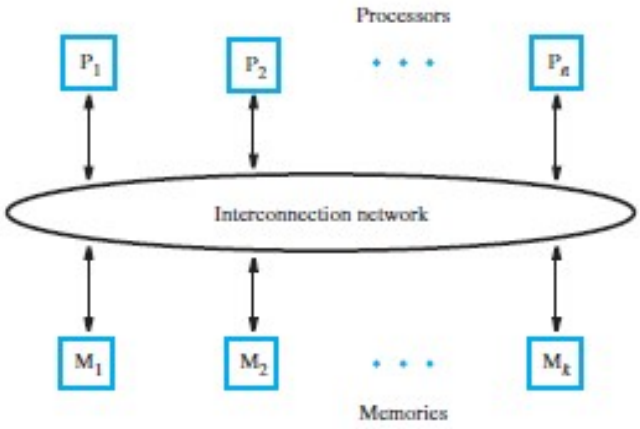


Figure 12.2 A UMA multiprocessor.

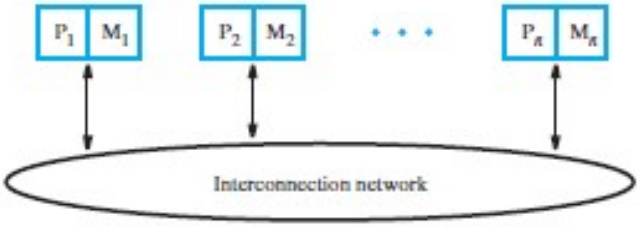


Figure 12.3 A NUMA multiprocessor.

•

In above two methods organization of memories is global, where any processor can access any memory module without intervention of another processor. A different way to organize the system is shown in figure 12.4. Here all memory modules serve as private memories for the processors that are directly connected to them. A processor cannot access a remote memory without the cooperation of the remote processor. This cooperation takes place in the form of message passing. Such systems are often called distributed memory systems with message passing protocol.

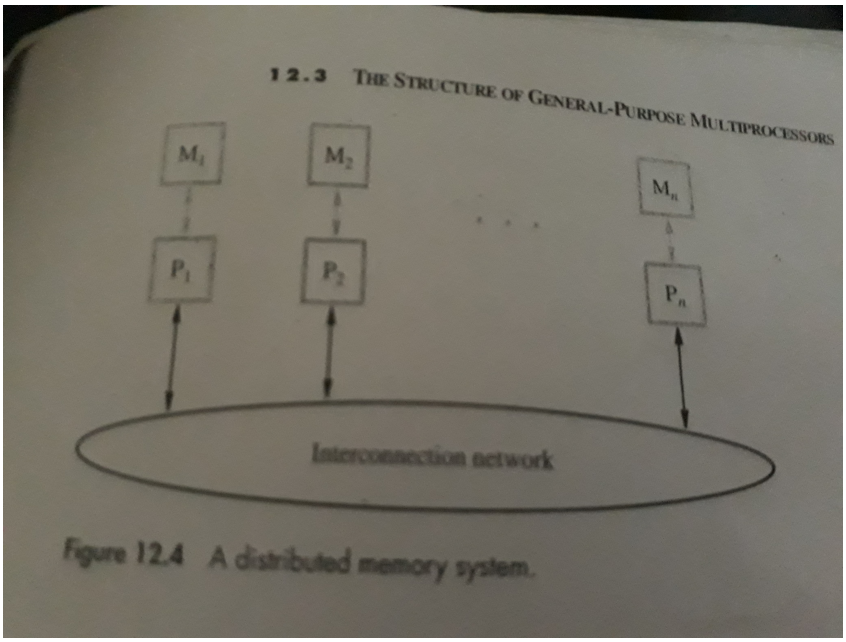


Figure 12.4 A distributed memory system.