UNIT I. REGISTER TRANSFER LOGIC AND CPU

Objectives

- To study what is register transfer and register transfer language. To study controlled Bus transfers and memory transfers. To understand the memory read and write operations.
- To learn what are micro operations and the various types of micro operations.
- To study the various organizations of the CPU and the various types of instructions.
- To understand the working of the control unit and the various cycles involved in the execution of an instruction.

INTRODUCTION

Digital systems are composed of modules that are constructed from digital components, such as registers, decoders, arithmetic elements, and control logic .The modules are interconnected with common data and control paths to form a digital computer system. Digital modules are defined by the registers they contain and the operations that are performed on the data stored in them. The operations performed on the data stored in the registers are called as micro operations.

Register transfer language is used to specify in symbolic form the sequence of micro operations that take place between the registers. There are register transfers and memory transfers in a computer system.

Operations performed on the data stored in the registers are called as micro operations. There are four types of micro operations: Register transfer micro operations, Arithmetic micro operations, Logic micro operations and Shift micro operations. These operations are implemented using various arithmetic and logic circuits.

There are three basic components of CPU: Register bank, ALU and Control Unit. There are several data movements between these units. The CPU has two types of organization: (i).General Register Organization and (ii). Stack Organization. Instruction formats are classified based on these organizations into zero address, one address, two address and three address instructions.

Control Unit causes the CPU to step through a series of micro-operations in proper sequence based on the program being executed. Processor fetches one instruction at a time and performs the operation specified. Purpose of control unit is to control the system operations by routing the selected data items to the selected processing HW at right time.

Basic computer operation cycle consists of the fetch cycle, decode cycle, indirect cycle, execute cycle and interrupt cycle.

REGISTER TRANSFER

Register Transfer Language (RTL) is

- A symbolic language
- A convenient tool for describing the internal organization of digital computers
- Can also be used to facilitate the design process of digital systems.

Register Transfer is

- Copying the contents of one register to another.

A register transfer is indicated as

R2 ← R1

In this case the contents of register R1 are copied (loaded) into register R2 .A simultaneous transfer of all bits from the source R1 to the destination register R2, during one clock pulse takes place. It should be noted that this is non-destructive; i.e. the contents of R1 are not altered by copying (loading) them to R2.

Control Functions

Often transfers occur only if a certain condition is true. This is similar to an "if" statement in a programming language. In digital systems, this is often done via a control signal, called a control function.

If the signal is 1, the action takes place. This is represented as: P: R2 ← R1

Which means "if P = 1, then load the contents of register R1 into register R2", i.e., if (P = 1) then (R2 ← R1).

Hardware Implementation of Controlled Transfers

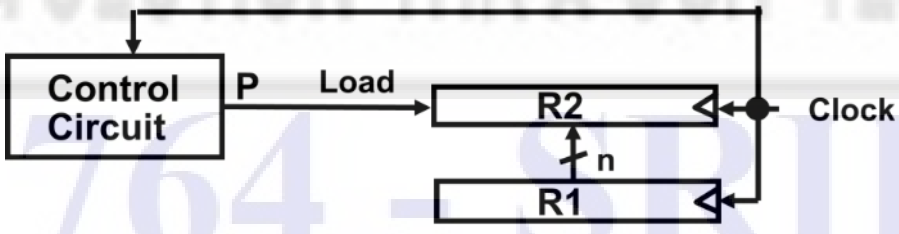


Figure 1.1 Controlled Transfers

- The same clock controls the circuits that generate the control function and the destination register
- Registers are assumed to use positive-edge-triggered flip-flops.

Bus And Bus Transfer

Bus is a path (of a group of wires) over which information is transferred, from any of several sources to any of several destinations.

A common bus system can be constructed using multiplexers. The multiplexers select the source register whose binary information is then placed on the bus.

The transfer of information from a bus into one of many destination registers can be accomplished by connecting the bus lines to the inputs of all destination register and activating the load control of the particular destination register selected.

Transfer from a register to bus (Fig 1.2): BUS ← R

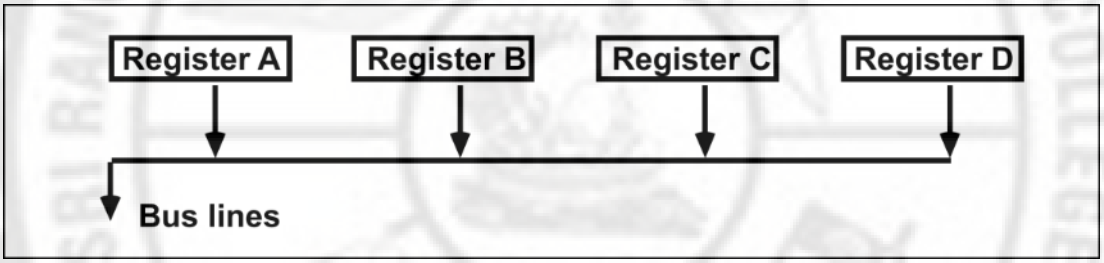


Figure 1.2 Transfer from a register to bus.

Bus Transfer In RTL

- Depending on whether the bus is to be mentioned explicitly or not, register transfer can be indicated as either
- R2 ← R1

Or

BUS ← R1, R2 ← BUS

In the former case the bus is implicit, but in the latter, it is explicitly indicated.

Memory and Memory Transfer

Memory (RAM) can be thought as a sequential circuits containing some number of registers These registers hold the words of memory. Each of the r registers is indicated by an address. These addresses range from 0 to r-1.Each register (word) can hold n bits of data. Assume the RAM contains r = 2k words. It needs the following

- n data input lines
- n data output lines
- k address lines
- A Read control line
- A Write control line

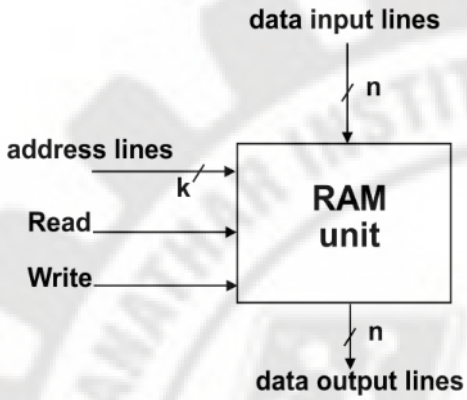


Figure 1.3 RAM

The memory is viewed at the register level as a device, M. Since it contains multiple locations, the address in memory must be specified when we access it. This is done by indexing memory references. Memory is usually accessed in computer systems by putting the desired address in a special register, the Memory Address Register (MAR, or AR). When memory is accessed, the contents of the MAR get sent to the memory unit's address lines.

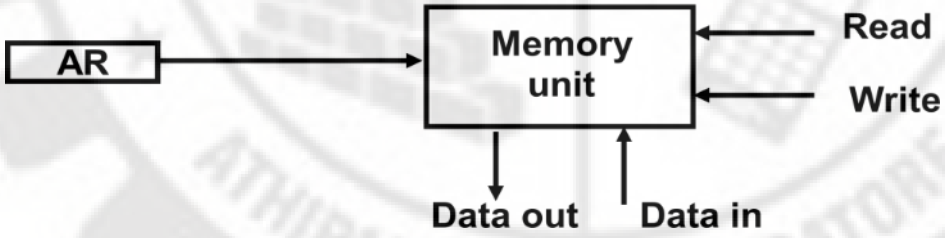


Figure 1.4 A Memory Unit with Read/Write control signals.

Memory Read Operation

• To read a value from a location in memory and load it into a register, the register transfer language notation is :

R1 ← M [MAR]

This causes the following to occur

- The contents of the MAR get sent to the memory address lines.
- A Read (= 1) gets sent to the memory unit.
- The contents of the specified address are put on the memory's output data lines.
- These get sent over the bus to be loaded into register R1.

Memory Write Operation

- To write a value from a register to a location in memory the register transfer language is

M [MAR] ← R1

This causes the following to occur

- The contents of the MAR get sent to the memory address lines.
- A Write (= 1) gets sent to the memory unit.
- The values in register R1 get sent over the bus to the data input lines of the memory.
- The values get loaded into the specified address in the memory.
-

MICRO OPERATIONS

Micro operation is an elementary operation performed (during one clock pulse), on the information stored in one or more registers. The functions built into registers are examples of micro operations :

- Shift
- Load
- Clear
- Increment

Computer system micro operations are of four types:

- Register transfer micro operations
- Arithmetic micro operations
- Logic micro operations
- Shift micro operations

Arithmetic Micro Operations:

The basic arithmetic micro operations are

- Addition

Example of addition:

R3 ←R1 +R2

- Subtraction

Subtraction is most often implemented through complementation and addition.

Example of subtraction: R3 ←R1 +~~R2~~+ 1.

Adding 1 to the 1's complement produces the 2's complement.

Adding the contents of R1 to the 2's complement of R2 is equivalent to subtracting.

- Increment
- Decrement

Multiply and divide are not included as micro operations . A micro operation is one that can be executed by one clock pulse. Multiply (divide) is implemented by a sequence of add and shift micro operations (subtract and shift).

Implementation of a Binary Adder-Subtractor Circuit

To implement the add microoperation with hardware, we need the registers that hold the data and the digital component that performs the addition . A full-adder adds two bits and a previous carry . A binary adder is a digital circuit that generates the arithmetic sum of two binary numbers of any length . A binary adder is constructed with full-adder circuits connected in cascade .An n-bit binary adder requires n full-adders .

The subtraction A-B can be carried out by the following steps

1. Take the 1's complement of B (invert each bit)
2 .Get the 2's complement by adding 1
3. Add the result to A
4. The addition and subtraction operations can be combined into one common circuit (Fig 1.5) by including an XOR gate with each full-adder.

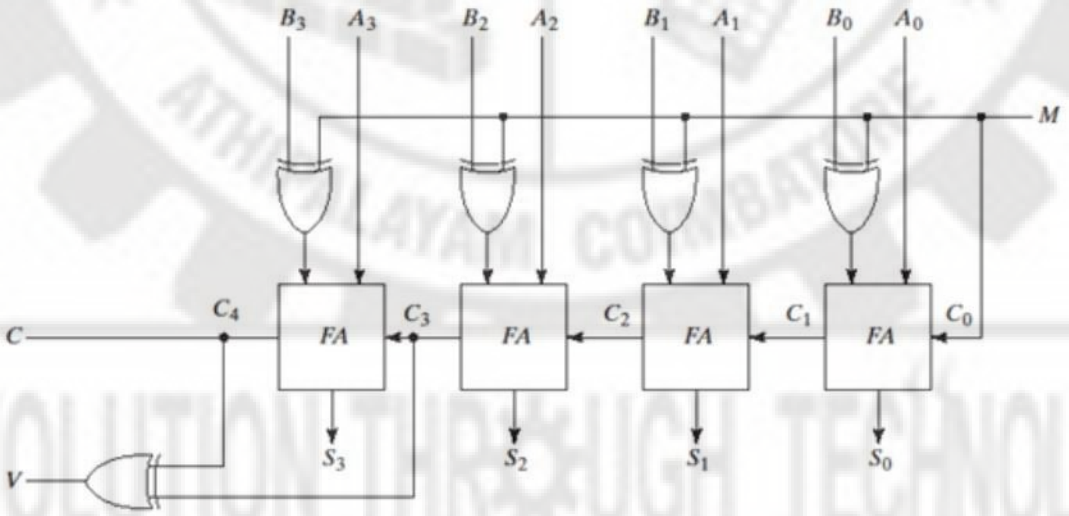


Figure 1.5 Four Bit Adder/Subtractor

Incrementer

The increment micro operation adds one to a number in a register .This can be implemented by using a binary counter – every time the count enable is active, the count is incremented by one. If the increment is to be performed independent of a particular register, then use half-adders connected in cascade . An n -bit binary incrementer requires n half-adders (Fig 1.6) .

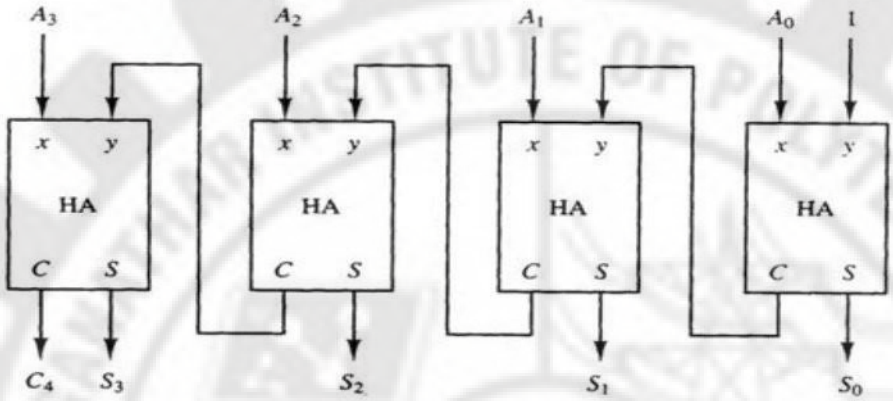


Figure 1.6 Four bit incrementer

2 -bit arithmetic circuit

An arithmetic circuit is a logic circuit that performs basic arithmetic operations like addition, subtraction, increment, decrement and transfer operations using a single combinational circuit.

It uses multiplexers and full adders to do so. There are two 4-bit inputs a and b and one 4 –bit output D. The selection bits, along with an additional input at 2 and 3 input positions of the multiplexer, determine the type of operation to be done on the input data.

The input a go directly to the X inputs of the binary adder .The inputs from b are connected to first data inputs of the multiplexers .The complements of b are connected to the second data inputs of the multiplexers .the other two data inputs of the multiplexers are connected to logic -0 and logic -1.The input carry C_{in} goes to the carry input of the first full adder .The other carries are connected from one stage to the next.

Output D= a + Y + C_{in}

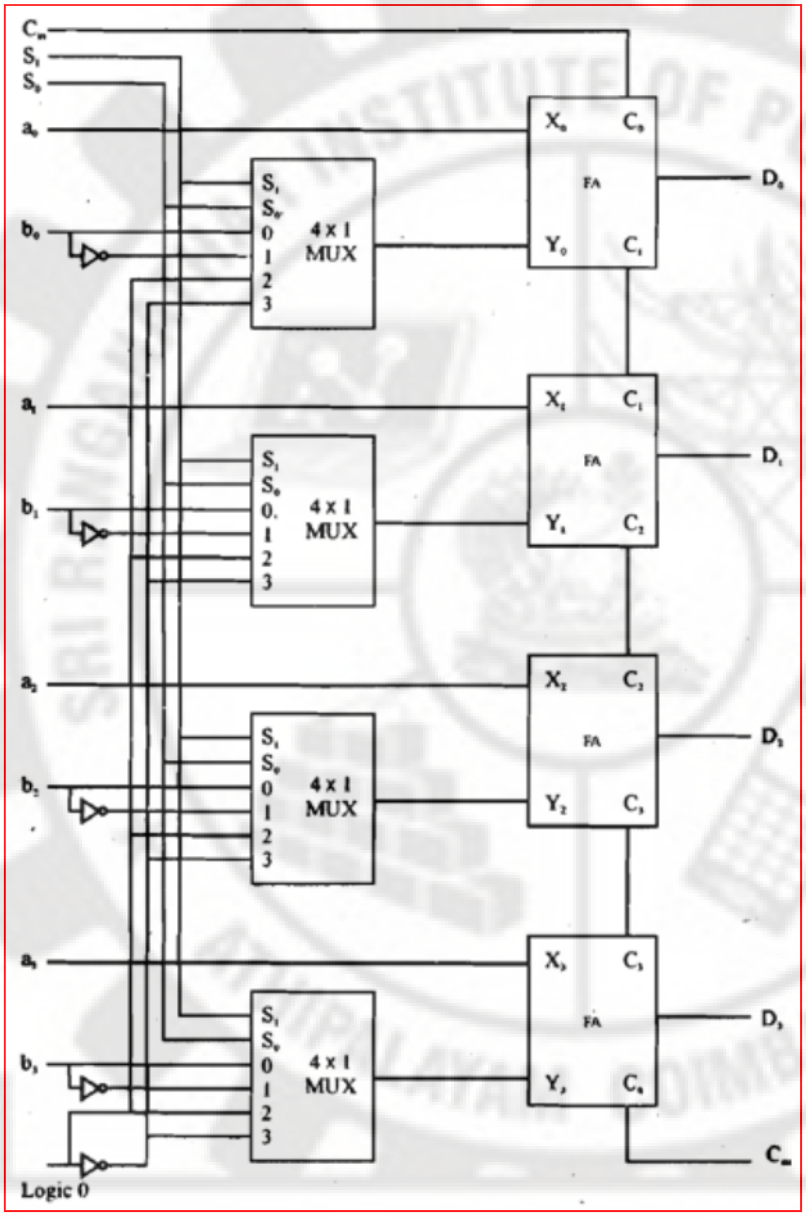


Figure 1.6.1 4 bit arithmetic circuit

Truth Table

Selection inputs			Input Y	FUNCTION $D=a+Y+C_{in}$	OPERATION
S1	S0	C_{IN}			
0	0	0	B	D=a+b	Addition
0	0	1	B	D=a+b+1	Addition with carry
0	1	0	b’	D=a+b’	Subtraction with carry
0	1	1	b’	D=a+b’+1	Subtraction
1	0	0	0	D=a	Transfer a
1	0	1	0	D=a+1	Increment a
1	1	0	1	D=a-1	Decrement a
1	1	1	1	D=a	Transfer a

Logic Micro Operations

Logic operations specify binary operations for strings of bits stored in registers and treat each bit separately.

Example: the XOR of R1 and R2 is symbolized by

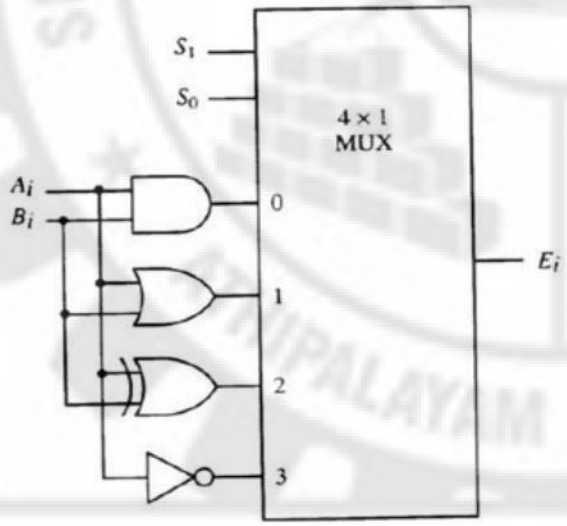
P: R1 ← R1 ⊕ R2

Example: If R1 = 1010 and R2 = 1100 then 0110 Content of R1 after P = 1

Symbols used for logical micro operations:

- OR: ∨
- AND: ∧
- XOR: ⊕

The + sign has two different meanings: logical OR and summation. When + is in a micro operation, then it is summation. When + is in a control function, then it is OR.



(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \bar{A}$	Complement

(b) Function table

Figure 1.7 One stage of Logic Circuit

Applications of Logic micro operations

Logic micro operations can be used to change bit values, delete a group of bits, or insert new bit values into a register .

- The selective-set operation sets to 1 the bits in A where there are corresponding 1's in B

1010 A before

1100 B (logic operand)

1110 A after

A ← A ∨ B

• The selective-complement operation complements bits in A where there are corresponding 1's in B

1010 A before

1100 B (logic operand)

0110 A after

$A \leftarrow A \oplus B$

• The selective-clear operation clears to 0 the bits in A only where there are corresponding 1's in B

1010 A before

1100 B (logic operand)

0010 A after

$A \leftarrow A \wedge B$

• The mask operation is similar to the selective-clear operation, except that the bits of A are cleared only where there are corresponding 0's in B

1010 A before

1100 B (logic operand)

1000 A after

$A \leftarrow A \wedge B$

• The insert operation inserts a new value into a group of bits

This is done by first masking the bits to be replaced and then Oring them with the bits to be inserted

0110 1010 A before

0000 1111 B (mask)

0000 1010 A after masking

0000 1010 A before

1001 0000 B (insert)

1001 1010 A after insertion

• The clear operation compares the bits in A and B and produces an all 0's result if the two number are equal

1010 A

1010 B

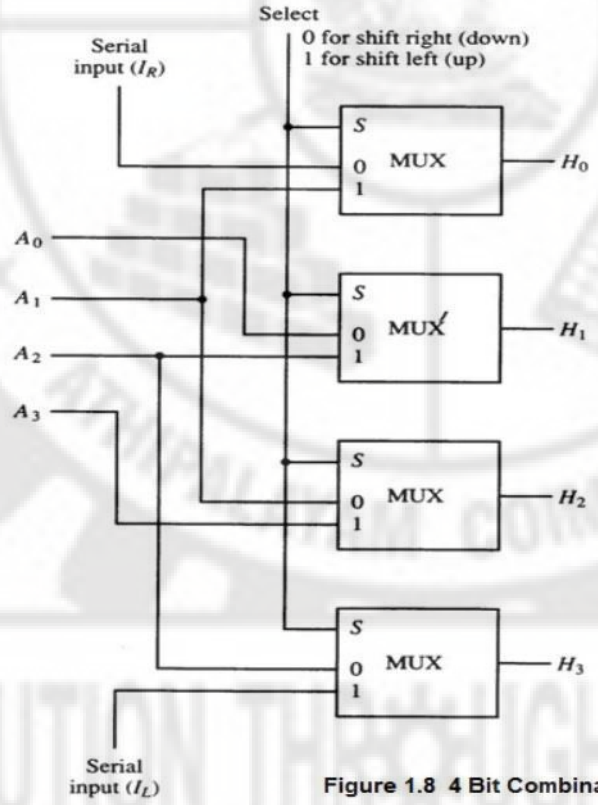
0000 $A \leftarrow A \oplus B$

Shift Micro Operations

Shift micro operations are used for serial transfer of data. They are also used in conjunction with arithmetic, logic, and other data-processing operations. There are three types of shifts: logical, circular, and arithmetic shift.

- A logical shift is one that transfers 0 through the serial input.The symbols shl and shr are for logical shift-left and shift-right by one position R1 ← shl R1.
- The circular shift (aka rotate) circulates the bits of the register around the two ends without loss of information. The symbols cil and cir are for circular shift left and right.

- The arithmetic shift shifts a signed binary number to the left or right. Shifting to the left is multiplying by 2, to the right is dividing by 2. Arithmetic shift leaves the sign bit unchanged. A sign reversal occurs if the bit in R_{n-1} changes in value after the shift. This happens if the multiplication causes an overflow. An overflow flip-flop V_s can be used to detect the overflow $V_s = R_{n-1} \oplus R_{n-2}$.



Function table

Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

Figure 1.8 4 Bit Combinatorial Circuit Shifter

A bi-directional shift unit with parallel load could be used to implement this. Two clock pulses are necessary with this configuration: one to load the value and another to shift. In a processor unit with many registers it is more efficient to implement the shift operation with a combinational circuit(Fig 1.8). The content of a register to be shifted is first placed onto a common bus and the output is connected to the combinational shifter, the shifted number is then loaded back into the register. This can be constructed with multiplexers.

Arithmetic Logic Unit

The arithmetic logic unit (ALU) is a common operational unit connected to a number of storage registers. To perform a microoperation, the contents of specified registers are

placed in the inputs of the ALU.The ALU performs an operation and the result is then transferred to a destination register. The ALU is a combinational circuit (Figure 1.9) so that the entire register transfer operation from the source registers through the ALU and into the destination register can be performed during one clock pulse period.

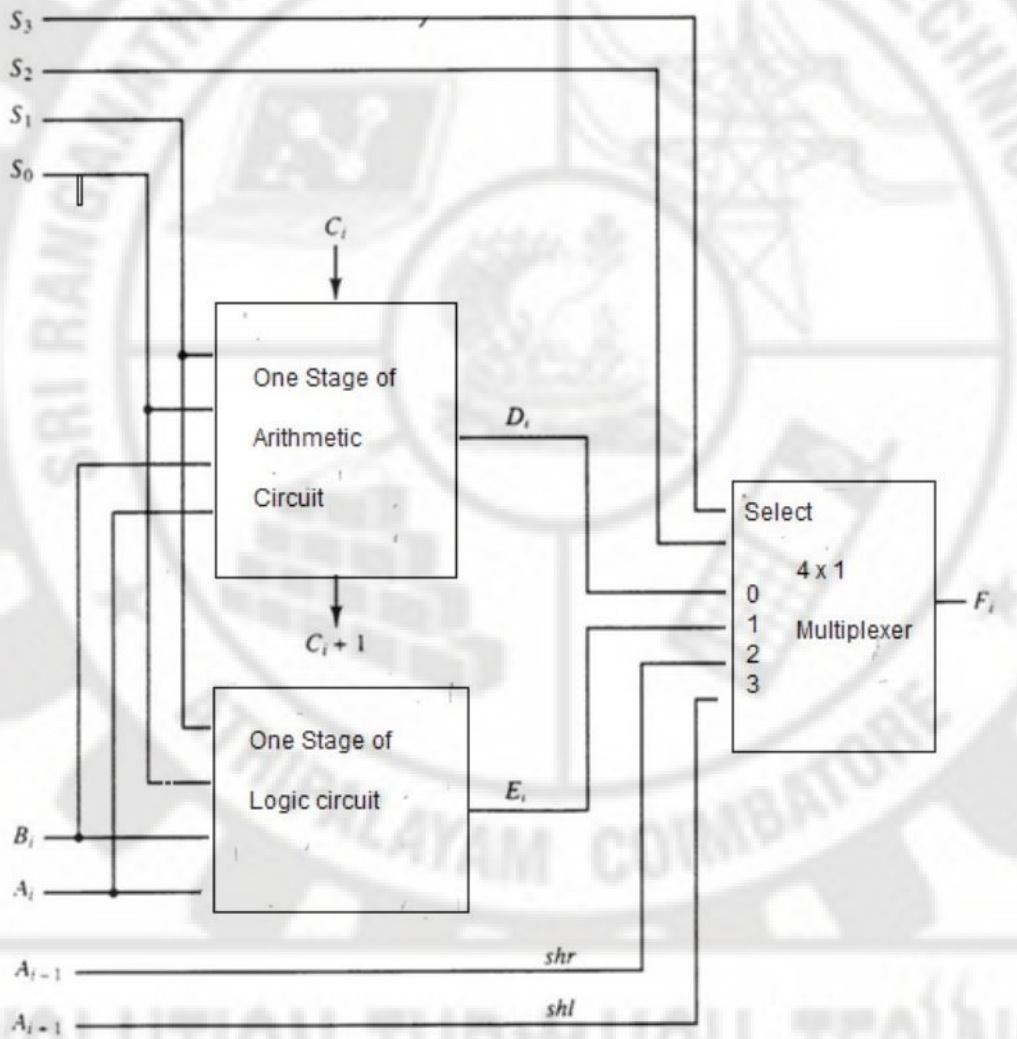


Fig 1.9 One Stage of Arithemtic and Logic Unit

CENTRAL PROCESSING UNIT

Components of CPU

The CPU (Figure 1.10)consists of :

1. Storage Components:

   Registers

   Flip-flops

2. Execution (Processing) Components:

Arithmetic Logic Unit (ALU) which performs Arithmetic calculations, Logical computations, Shifts/Rotates.

3. Transfer Components: Bus

4. Control Components: Control Unit

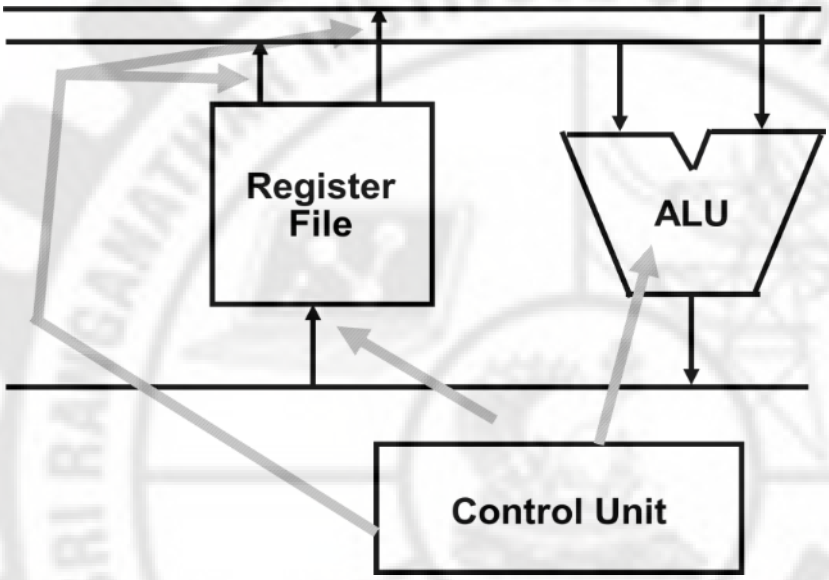


Figure 1.10 CPU Components

General Register Organization

In Basic Computer, there is only one general purpose register, the Accumulator (AC). In modern CPUs, there are many general purpose registers. It is advantageous to have many registers. Transfer between registers within the processor is relatively fast.

These registers are connected through a common bus. The registers communicate with each other not only for direct data transfers, but also while performing various micro-operations. A bus organization for seven registers is shown in figure 1.11. The output of each register is connected to two multiplexers (MUX) to form the two buses A and B. The selection lines in each multiplexer select one register or the input data for the particular bus. The A and B buses form the inputs to a common arithmetic logic unit (ALU). The operation selected in the ALU determines the arithmetic or logic micro operation that is to be performed. The result of the micro operation is available for output data and also goes into the inputs of all the registers. The register that receives the information from the output bus is selected by a decoder. The decoder activates one of the register load inputs, thus providing a transfer path between the data in the output bus and the inputs of the selected destination register.

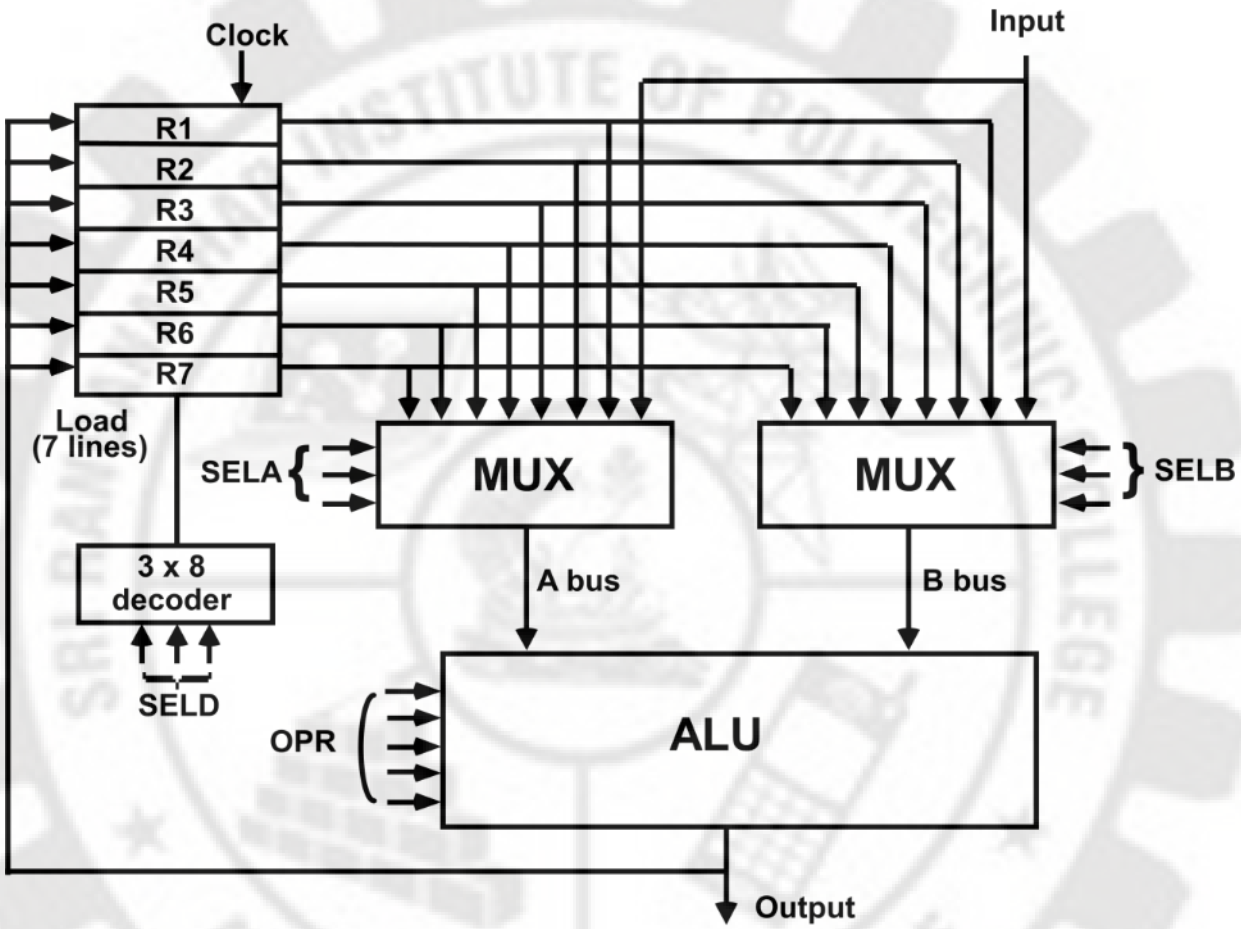


Figure 1.11 General Register Organizations

The control unit directs the information flow through ALU by

i. Selecting various Components in the system

ii. Selecting the Function of ALU

Example: R1 ← R2 + R3

1] MUX A selector (SELA): BUS A ←R2

[2] MUX B selector (SELB): BUS B ← R3

[3] ALU operation selector (OPR): ALU to ADD

[4] Decoder destination selector (SELD): R1 ← out Bus

Register Stack Organization

Stack is a very useful feature that is included in the CPU of most computers. This can be accessed as a Last In First Out list (LIFO).The stack in digital computers is a memory unit with a stack pointer (SP).The two operations that can be performed on a stack are the insertion and deletion of items. The operation of insertion is called push because it results in pushing the new item on top of the stack. The operation of deletion is called pop because it results in removing the item from the top of the stack. A stack can be placed in a portion of a large memory or it can be organized as a collection of finite number of registers. Figure 1.12

shows the organization of a 64 word register stack. The stack pointer register contains the address of the word that is currently on top of the stack. Three items are placed in the stack: A, B, and C. Item C is on top of the stack so that the content of SP is 3.To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the SP by one so that it holds 2.To insert a new item, the push operation is carried out by incrementing SP and writing the new word in that location pointed to by the stack.

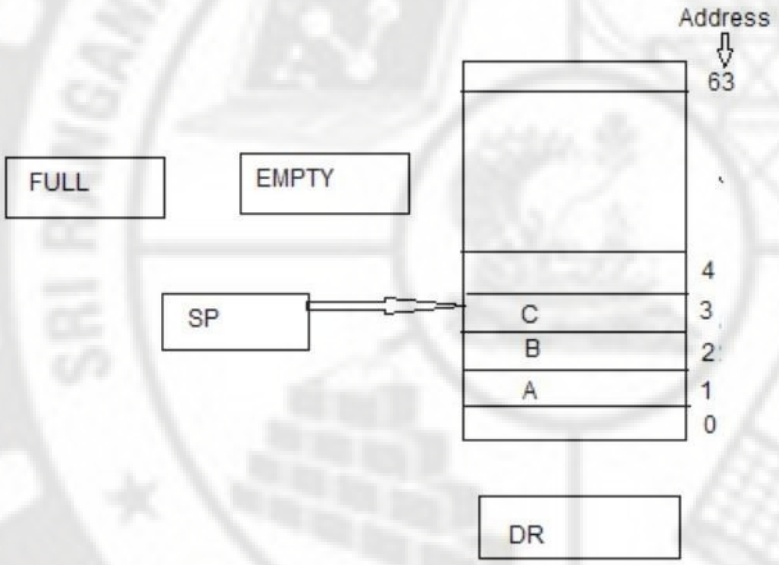


Fig 1.12 Computer Stack Organization using 64 bit words

Push, Pop operations

/* Initially, SP = 0, EMPTY = 1, FULL = 0 */

PUSH	POP
SP ¬ SP + 1	DR ¬ M [SP]
M[SP] ¬ DR	SP ¬ SP – 1
If (SP = 0) then (FULL ¬ 1)	If (SP = 0) then (EMPTY ¬ 1)
EMPTY ¬ 0	FULL ¬ 0

Memory Stack

A stack can exist as a stand alone unit as in fig. 1.12 or can be implemented in a random access memory attached to a CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. Fig 1.13 shows a portion of computer memory partitioned into three segments: Program, Data and stack. The program counter PC points at the address of the next instruction in the program. The address register AR points at an array of data. The stack pointer points to the top of the stack.

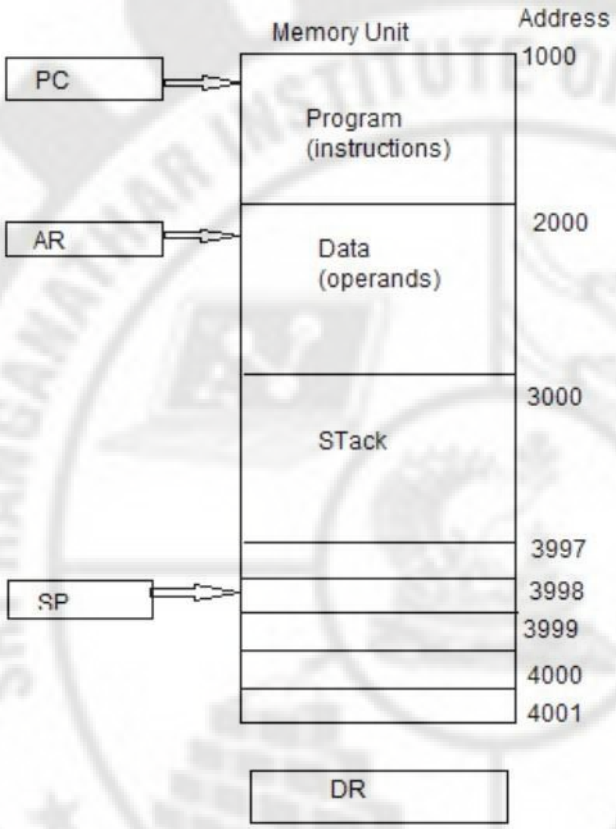


Fig 1.13 Computer Memory with Program,stack and code segments

The three registers are connected to a common address bus, and either one can provide an address for memory. PC is used during the fetch phase to read an instruction.AR is used during the execute phase to read an operand. SP is used to push or pop items into or from the stack. As shown in fig.1.13 the initial value of SP is 4001 and the stack grows with decreasing addresses. Thus the first item stored in the stack is at address 4000, the second item is stored at address 3999 and the last address that can be used for the stack is 3000. No provisions are available for stack limit checks.

Stack Operations:

The items in the stack communicate through a data register DR.

PUSH

A new item is inserted with the push operation as follows:

SP ← SP – 1

M[SP] ← DR

The stack pointer is decremented by one and the contents of the DR register are stored in the memory location pointed to by the SP.

POP

A new item is removed from the stack as follows:

DR ← M[SP]

SP SP – 1 ←

The contents of the stack is stored in the DR register. The stack pointer is incremented by 1.

Stack Limits

There is no method provided by the computers to check underflow or overflow of the stack. The stack limits can be checked by using two processor registers one to store the upper limit of the stack and the other one to store the lower limit of the stack. After a push operation the SP is compared with the upper limit register and after a pop operation the SP is compared with the lower limit register. The advantage of a memory stack is that the CPU can refer to it without having to specify an address, since the address is always available.

Instruction Format

The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register. The bits of the instruction are divided into groups called fields. The most common fields found in instruction formats are:

- OP-code field - specifies the operation to be performed
- Address field - designates memory address(es) or a processor register(s)
- Mode field - determines how the address field is to be interpreted (to get effective address or the operand)

The operation code field defines the various operations such as add,subtract,complement and shift. The bits that define the mode field of an instruction code specifies a variety of alternatives for choosing the operands from the given address. Operations specified by computer instructions are specified by their memory address. Operands residing in memory are specified by their memory address. Operands residing in processor registers are specified with a register address.

The number of address fields in the instruction format depends on the internal organization of CPU. The three most common CPU organizations are:

(i) Single accumulator organization:

Here all operations are performed with the help of a accumulator register. For example the instruction that specifies an arithmetic addition is defined as

ADD X /* AC ← AC + M[X] */

Here AC is the accumulator register and M[X] is the contents of the data stored at the location X. All operations are performed between the accumulator and the contents of the memory operand.Single address instructions fall in this category.

Single Address instructions

Program to evaluate X = (A + B) * (C + D) :

LOAD A /* AC ← M[A] */

ADD B /* AC ← AC + M[B] */

```
STORE     T      /*  M[T] ← AC      */
LOAD      C      /* AC ← M[C]       */
ADD       D      /* AC ← AC + M[D]  */
MUL       T      /* AC ← AC * M[T]  */
STORE     X      /*  M[X] ← AC      */
```

(ii)General register organization:

The instruction format in this type of computer needs three address fields. The three address fields can be either registers or memory locations. Three address instructions belong to this category. Thus the instruction for an arithmetic addition is

```
ADD   R1, R2, R3      /* R1 ← R2 + R3  */
```

Three Address Instructions

Program to evaluate X = (A + B) * (C + D) :

```
ADD     R1, A, B      /*  R1      M[A] + M[B]   */
ADD     R2, C, D      /*  R2      M[C] + M[D]   */
MUL   X, R1, R2       /*  M[X]    R1 * R2       */
```

Advantage

Results in short programs

Limitations

Instruction becomes long (many bits)

Two-Address Instructions

The number of address fields can be reduced to two if the destination register is the same as one of the source registers. Here again the two address fields can be either registers or memory addresses.

```
ADD   R1, R2          /* R1 ← R1 + R2 */
MOV   R1, R2          /* R1 ← R2  */
ADD   R1, X           /* R1 ← R1 + M[X]  */
```

Program to evaluate X = (A + B) * (C + D) :

```
MOV   R1, A       /* R1   M[A]      */
ADD   R1, B       /* R1   R1 + M[A]  */
MOV   R2, C       /* R2   M[C]      */
ADD   R2, D       /* R2   R2 + M[D] */
MUL   R1, R2      /* R1   R1 * R2     */
MOV   X, R1       /* M[X]   ← R1      */
```

Stack organization:

The computers with stack organization will have Push and Pop instructions which require an address. Operation type instructions do not require any addresses. The Add instruction given below performs the add operation by popping the two numbers from the top of the stack and storing the result again on top of the stack. Zero address instructions belong to this category.

```
PUSH    X              /* TOS ← M[X] */
ADD
```

Zero-Address Instructions

Program to evaluate X = (A + B) * (C + D) :

```
PUSH A      /* TOS ← A          */
PUSH B      /* TOS ← B          */
ADD         /* TOS ← (A + B)    */
PUSH C      /* TOS ← C          */
PUSH D      /* TOS ← D          */
ADD         /* TOS ← (C + D)    */
MUL         /* TOS ← (C + D) * (A + B)  */
POP  X      /* M[X] ← TOS       */
```

CONTROL UNIT

Structure of Control Unit

Purpose of control unit is to control the system operations by routing the selected data items to the selected processing HW at right time. Control unit's responsibility is to drive the associated processing HW by generating a set of signals that are synchronized with the master clock. In order to carry out a task such as ADD, The control unit must generate a set of control signals in a predefined sequence governed by the HW structure of the processing section.

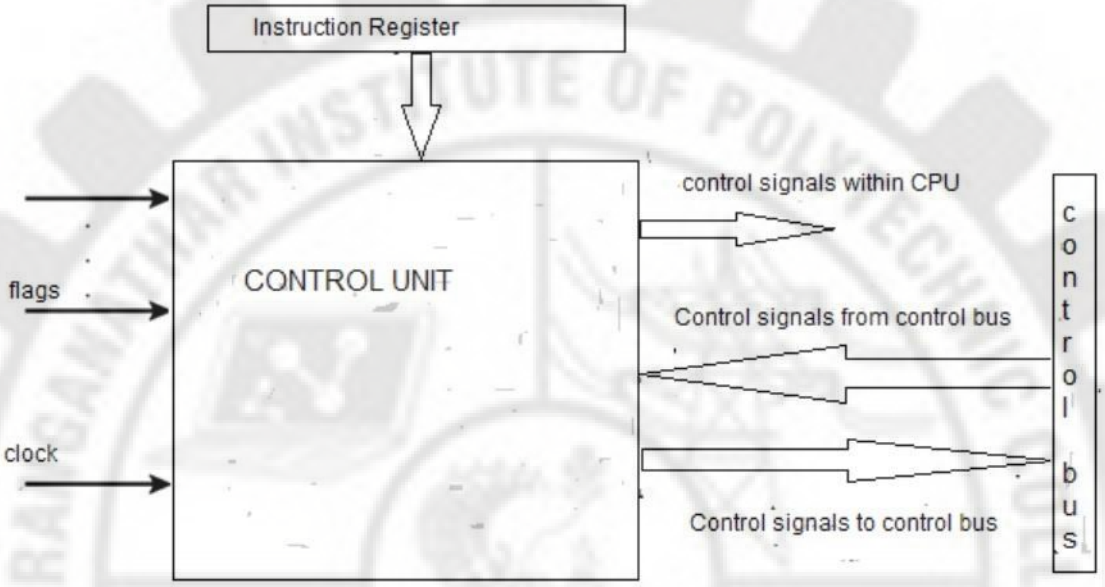


Fig 1.14 Structure of Control Unit

Functions of the Control Unit

- Sequencing
- CU causes the CPU to step through a series of micro-operations in proper sequence based on the program being executed.
- Execution
- CU causes each micro-operation to be performed.

Control Signals

- External signals are the inputs indicating the state of the system.
- Internal signals are the logics required to perform the sequencing and execution functions.

Inputs to control unit are:

- Master clock
- Status info from processing section
- Command signals from external agent.

Outputs produced by control unit are:

- Signals that drive the processing section and responses to an external environmentt (operation complete or abort) due to exceptions (overflow and underflow).

Control unit undertakes the following responsibilities

- Instruction interpretation: (read instr. , recognize, get operands and route to appropriate functional units, necessary control signals issued)
- Instruction sequencing: control unit determines the address of next instruction to be executed and loads to PC
-

Instruction Cycle

A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle for each instruction called as instruction cycle. The instruction cycle in turn is subdivided into a sequence of subcycles or phases. An instruction cycle consists of the following phase:

1. Fetch an instruction from memory
2. Decode the instruction
3. Read the effective address from memory if the instruction has an indirect address
4. Execute the instruction until a HALT instruction is encountered

In general, there is a sequence of events that can be described that make up the execution of an instruction

- Fetch cycle
- Data fetch cycle
- Indirect cycle
- Execute cycle
- Interrupt cycle

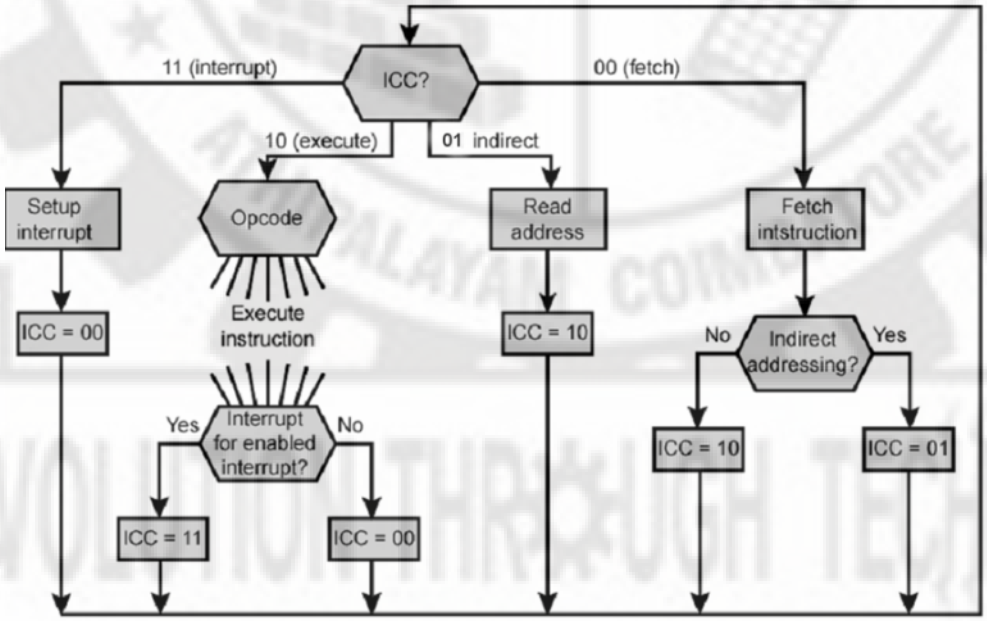


Figure 1.15 Flowchart for instruction Cycle

Control Registers used during the execution of an instruction are

- Program Counter (PC)
  - Holds address of next instruction to be fetched
- Memory Address Register (MAR)
  - Connected to address bus
  - Specifies address for read or write op
- Memory Buffer Register (MBR)
  - Connected to data bus
  - Holds data to write or last data read

- Instruction Register (IR)

— Holds last instruction fetched

—

Fetch Cycle:

Two fetch operations are present in an instruction fetch cycle: Instruction fetch and Data fetch.

Instruction Fetch

PC contains address of next instruction. The various steps involved are :

1. Address moved to Memory Address Register (MAR).
2. Address placed on address bus
3. Control unit requests memory read.
4. Result placed on data bus, copied to Memory Buffer Register (MBR), then to IR.
5. Meanwhile PC incremented by size of machine code (typically one address).

1. t1: MAR ← (PC)
2. t2: MBR ← (memory)
3. PC ← (PC) +1
4. t3: IR ← (MBR

Data Fetch

1. Operand address is fetched into MBR
2. IR is examined to determine if indirect addressing is needed. If so, indirect cycle is performed:
   a. Address of location from which to fetch operand address is calculated based on first fetch
   b. Control unit requests memory read
   c. Result (actual address of operand) moved to MBR
3. Address in MBR moved to MAR
4. Address placed on address bus
5. Control unit requests memory read
6. Result placed on data bus, copied to MBR

Indirect Cycle

Some instructions require operands, each of which requires a memory access. With indirect addressing, an additional memory access is required to determine final operand address. Indirect addressing may be required of more than one operand, e.g., a source and a destination. Each time indirect addressing is used, an additional operand fetch cycle is required.

Indirect Cycle:

1. t1: MAR ← ($IR_{address}$)
2. t2: MBR ← (memory)
3. t3: $IR_{address}$ ← ($MBR_{address}$)

Execute Cycle

• Due to wide range of instruction complexity, execute cycle may take one of many forms.

– register-to-register transfer
– memory or I/O read
– ALU operation

Execute Cycle: ADD R1, X

• t1: MAR ← ($IR_{address}$)
• t2: MBR ← (memory)
• t3: R1 ← R1 + (MBR)

Interrupt Cycle

At the end of the execution of an instruction, interrupts are checked. Unlike execute cycle, this cycle is simple and predictable. If no interrupt pending

– go to instruction fetch

If interrupt pending

– Current PC saved to allow resumption after interrupt
– Contents of PC copied to MBR
– Special memory location (e.g. stack pointer) loaded to MAR
– MBR written to memory
– PC loaded with address of interrupt handling routine
– Next instruction (first of interrupt handler) can be fetched

Interrupt Cycle:

1. t1: MBR ←(PC)
2. t2: MAR ← save-address
   PC ← routine-address
3. t3: memory ← (MBR)
4.

Summary:

The Register Transfer Language(RTL) for the various microoperations have been discussed. The memory transfer and bus transfer operations have been discussed in detail. Memory read and memory write operations have also been explained. The concept of a micro operation and the types of micro operations have been discussed. The implementation of these using hardware is also explained. The various CPU organizations and the instruction types based on these have been explained. The main role of the control unit and the various steps of the instruction cycle have also been explained in detail.

Review Questions

2 marks questions

1) What is RTL.
2) What is a control function.
3) What is a bus.
4) What is a memory.
5) What is a memory read operation.
6) What is a memory write operation.
7) What is a micro operation.
8) What are the different types of micro operations.
9) List down the various arithmetic micro operations
10) List down the various shift micro operations.
11) List down the various logical micro operations.
12) What are the various components of a CPU.
13) What are the various organizations of a CPU.
14) What are the various fields in an instruction format.
15) What is the significance of the opcode field.
16) What are zero address instructions.
17) Give example for single address and two address instructions.
18) What is a control unit.
19) Draw the structure of a Control unit.
20) List down the functions of a control unit.
21) What is an instruction cycle.
22) What are the various phases in an instruction cycle.

3 marks Questions

1) Explain register transfer using RTL.
2) Explain controlled transfer with the help of a block diagram.
3) Give the RTL for memory read operation and explain the same.
4) Give the RTL for memory write operation and explain the same.
5) Explain about the various arithmetic operations in detail.
6) Explain about the various shift microoperations.
7) Explain the various logical microperations in detail.
8) Draw the block diagram of a general register organization of a CPU.
9) Explain the Push and Pop operation in a memory stack.
10) Explain the push and pop operation in a register stack.
11) What is stack limit. Explain.
12) Explain in detail the various fields in the Instruction Format.

13) Explain the structure of a control unit in detail.
14) Explain the fetch cycle in detail.
15) Explain the indirect cycle in detail.
16) Explain the interrupt cycle in detail.

5 / 10 marks Questions

1) Explain What is a bus and bus transfer with a neat block diagram.
2) Explain memory read and write operation in detail with neat block diagrams.
3) What is a control function. Explain controlled Transfer in detail.
4) Explain the functioning of a single stage ALU with a neat block diagram.
5) Explain the operation of a logical shifter with a neat diagram.
6) Explain the operation of a logical unit with a neat block diagram.
7) Explain the General Register Organization of a CPU with block diagram.
8) Explain with block diagram the register stack organization in detail.
9) Explain in detail the memory stack organization with block diagram.
10) Explain the Zero address, One, Two and Three address instructions with example.
11) Explain in detail the functioning of the control unit with block diagram.
12) Explain the fetch cycle in detail.
13) Explain the Indirect and Interrupt cycle in detail.
14) Explain the execute cycle with example.

UNIT II. INPUT /OUTPUT MODULES

Objectives

- To know the Connection of input output units with the CPU .
- To know about different asynchronous data transfer methods, the signals used and timing diagrams and to know the interface IC's for asynchronous transfer.
- To understand the different modes of data transfer, their advantages and disadvantages and to know the interface IC's for Interrupt and DMA operations.
- To understand IOP processor , communication between CPU and IOP

2.0 INTROUDUCTION

One of the basics features of a computer is its ability to exchange data with other devices. A computer has to get data from various devices and to give the result to various devices. Input output interfaces are required for solving the difference in the devices. i.e., signal format, mode of transfer etc with the CPU. The transfer may be synchronous or asynchronous method of data transfer between two devices when they are placed apart. In asynchronous mode special control signals are required like strobe and handshaking. Different modes of data transfer between the input output and memory are happening like programmed I/O, interrupt initiated I/O and DMA data transfer. IOP the Input output processor are used for interface. The data transfer is also in serial form and different modes of serial communication are there. (i.e) simplex, half duplex and full duplex mode. In the following chapters we are going to discuss it in detail.

INPUT OUTPUT INTERFACE

Peripherals connected to a computer need special communication link for interfacing them with the CPU. The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral. i.e it provides a method for transferring information between internal storage and external I/O devices. The communication link is known as Input Output interface..

Need for Interface Unit

- Peripherals are electromechanical and electromagnetic devices and their operation is different from the operation of the CPU and memory which are electronic devices .therefore conversion of signal values may be required.
- The data transfer rate of peripherals is usually slower than the transfer rate of the CPU,so synchronization mechanism is needed.
- Data codes and formats in peripherals differ from the word format in the CPU and memory

- The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

Therefore the main function of input output interface circuit should be data conversion, synchronization and device selection. Data conversion refers to conversion between digital and analog signals and conversion between serial and parallel data formats. Synchronization refers to matching of operating speeds of CPU and other peripherals. Device selection refers to the selection of I/O device by CPU in a queue manner.

I/O Bus and Interface Modules

A typical communication link between the processor and several peripherals is shown in Fig 2.1 .The I/O bus consists of data lines, address lines and control lines. Each peripheral device has associated with it an interface unit. Each interface decodes the address and control received from the I/O bus, interprets them for the peripheral, and provides signals for the peripheral controller. It also synchronizes the data flow and supervises the transfer between peripheral and processor. Each peripheral has its own Controller that operates the particular electro mechanical device.

The I/O bus from the processor is attached to all peripheral interfaces. To communicate with a particular device the processor places a device address on the address lines. Each interface attached to the I/O bus contains an address decoder that monitors the address lines. When the interface detects its own address it activates the path between the bus lines and the device it controls. All peripherals whose address does not corresponds to the address in the bus are disabled by their interface.

I/O Commands

When the address is made available in the address lines the processor provides a function code in the control lines. The function code is known as I/O command. There are four types of commands

- Control
- Status
- Data output and
- Data input

Control command: issued to activate the peripheral and to inform it what to do

Status command: used to test various status conditions in the interface and the peripheral.

Data output command: causes the interface to respond by transferring data from the bus into one of its registers

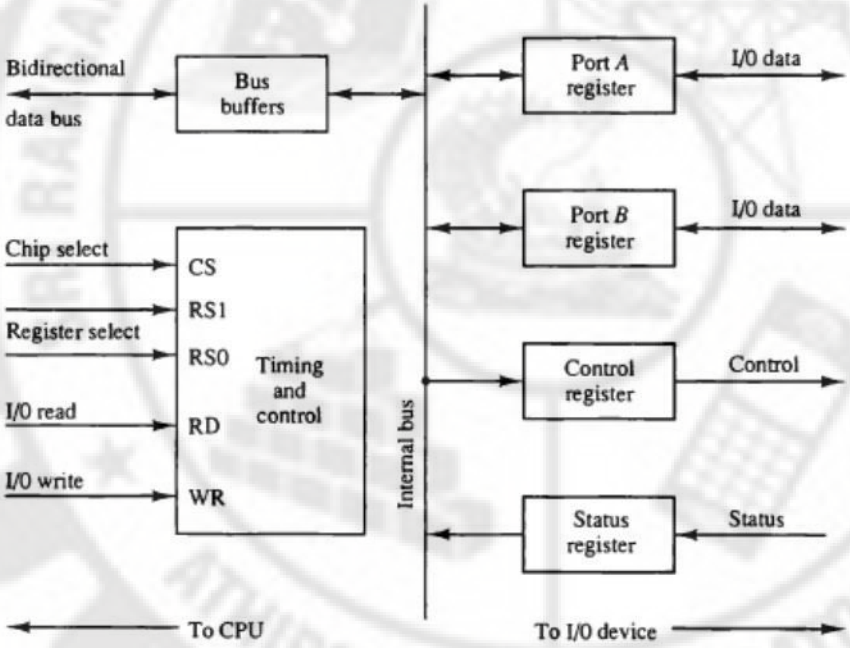
Data input command: the interface receives an item of item from the peripheral and places it in its buffer register.

I/O Interface

An example of an I/O interface unit is shown in block diagram in fig 2.2. It consists of two data registers called *ports ,* a control register, a status register, bus buffers and timing

and control circuits. The interface communicates with the CPU through the data bus. The chip select and register select inputs determine the address assigned to the interface. The I/O read and write are two control lines that specify an input or output respectively. The four registers communicate directly with the I/O device attached to the interface.

The I/O data to and from the device can be transferred into either port A or port B. The interface may operate with an output device or with an input device or with a device that requires both input and output.



CS	RS1	RS0	Register selected
0	×	×	None: data bus in high-impedance
1	0	0	Port *A* register
1	0	1	Port *B* register
1	1	0	Control register
1	1	1	Status register

Fig 2.2 Example of I/O interface unit.

The control register receives control information from the CPU. By loading appropriate bits into the control register, the interface and the I/O device attached to it can be placed in a variety of operating modes. The bits in the status register are used for status conditions and for recording errors that may occur during the data transfer.

The interface register communicate with the CPU through the bidirectional data bus. The address bus selects the interface unit through the chip select and two register select inputs. A circuit must be provided externally to detect the address assigned to the interface registers. this circuit enables the chip select input when the interface is selected by the address bus. The register select lines and are usually connected to the two least significant lines of the address bus. These two inputs select one of the four registers in the interface as

shown in the table. The content of the selected register is transfer into the CPU via data bus when the I/O read signal is enabled. The CPU transfers binary information into the selected register via the data bus when the I/O write input is enabled.

ASYNCHRONOUS DATA TRANSFER

If the registers in the interface share a common clock with the CPU registers, the transfer between the two units is said to be synchronous. In most cases the internal timing in each unit is independent from the other in that each uses its own private clock for internal registers. In that case the two units are said to be asynchronous to each other.

In this type control signals be transmitted between the communicating devices to indicate the time at which data is be transmitted. Strobe is the method using the control pulse supplied by one of the units to indicate to the other unit when the transfer has to occur. Handshaking is the method where the unit receiving the data item responds with another control signal to acknowledge receipt of the data.

Strobe Control Method

A single control line that informs the other end of the unit about whether data bus contains valid data on the bus or request for a data on the data bus is known as strobe control method. The communication are of two types, source initiated and destination initiated.

Source Initiated: The fig 2.3 show the block diagram and timing diagram

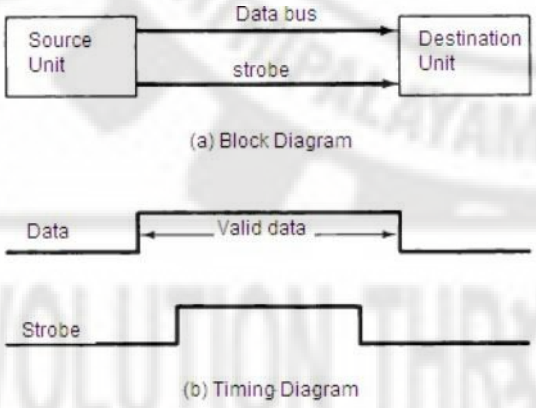


Fig 2.3 Source initiated Strobe for data transfer

The following steps are happening

- Source unit places data in the data bus
- After a delay to ensure settlement of data on the bus strobe pulse is initiated.
- After for a sufficient time, during the falling edge of the strobe pulse transfer the data from bus into internal register.
- Source removes the data from the data bus.
- New valid data will be available only after the strobe is enabled again

2.2.1.2.Destination Initiated: The fig 2.4 show the block diagram and timing diagram and the following steps are happening

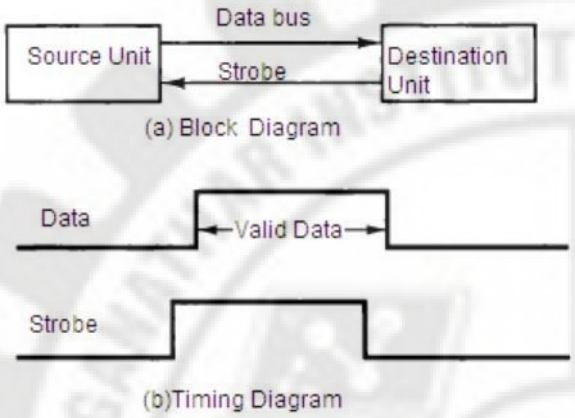


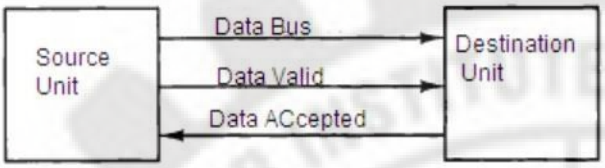
Fig 2.4 Destination initiated strobe control

- Destination unit activates strobe pulse informing source to provide data
- Source place data on the data bus
- The data remain in the bus long enough for the destination unit to accept it
- The falling edge of the strobe used again to trigger the destination register
- The destination unit disables the strobe.
- The source removes the data from the bus after a predetermined time interval
- *Disadvantage of Strobe*
- Source initiated does not know whether the destination unit actually received the data
- Destination initiated does not know whether the source actually placed the new data in the data bus.

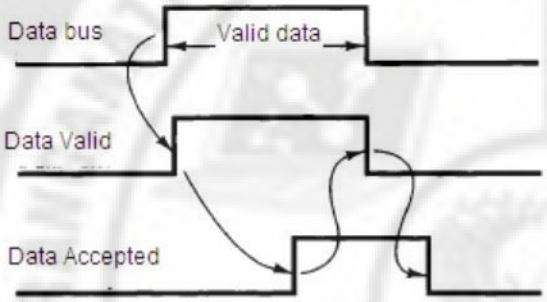
Hand Shaking Method

The two wire control method, one control line is in the same direction as the data flow in the bus from the source to destination.(valid data). The other control line is in the other direction from the destination to the source.(data accepted/ready for data). The sequence of control during the transfer depends on the unit that initiated the transfer.

Source Initiated Handshaking: The fig 2.5 show the block diagram and timing diagram and the following steps are happening:



(a) Block Diagram



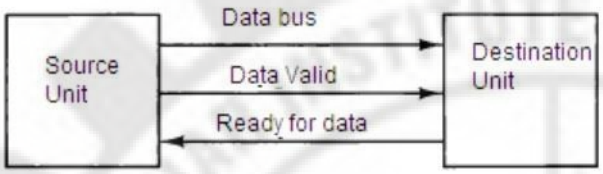
(b) Timing Diagram

Fig 2.5 Source initiated transfer using Handshaking

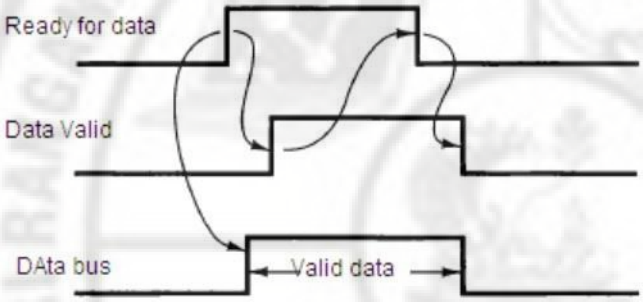
- Source place data on the data bus
- Data valid signal initiated by the source
- Data accepted signal initiated by destination after receives the data from the bus
- Data valid signal is disabled
- Data accepted signal is disabled.
- After data accepted signal is disabled the source is ready to send next data

Destination Initiated Handshaking: The fig 2.6 show the block diagram and timing diagram and the following steps are happening

- Destination place ready for data signal to source
- Source place data on the data bus
- Data valid signal is initiated by the source.
- Ready for data signal is disabled after receives the data from the bus
- Data valid signal is disabled.
- Invalidate the data on the bus.



(a) Block Diagram



(b) Timing Diagram

Fig 2.6 DEstination initiated transfer for handshaking

2.2.3.Asynchronous Serial Transfer:

The transfer of data between two units may be done in parallel or serial. In parallel data transmission n bit message must be transmitted through n separate conductor paths at the same time. In serial data transmission each bit in the message is sent in sequence one at a time. Serial transmission is slower but is less expensive since it requires only one pair of conductors.

In asynchronous transmission binary information is sent only when it is available and the line remains idle when there is no information to be transmitted.

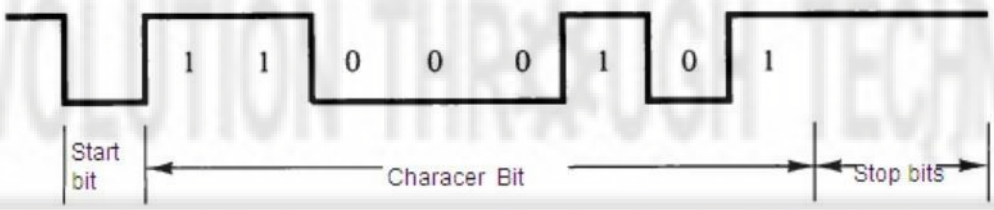


Fig 2.7 Asynchronous Serial Transmission

A serial asynchronous data transmission technique used in many interactive terminals use special bits that are inserted at both ends of the character code. Each character consists of three parts. A start bit, character bits and stop bits.

The transmitter rests is normally in 1 state. The first bit called the start bit always a 0 and is used to indicate the beginning of a character. The last bit called the stop bit is always a 1 . An example of this format is shown in fig 2.7

A transmitted character can be detected by the receiver from knowledge of the transmission rules

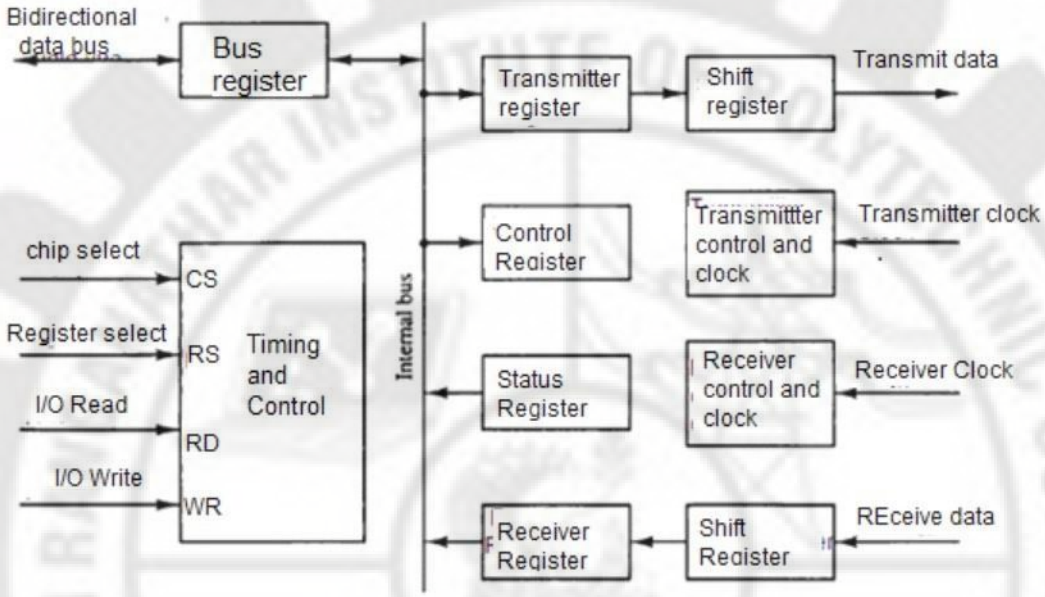
- When a character is not being sent, the line is kept in the 1 state
- The initiation of a character transmission is detected from the start bit which is always 0
- The character bits always follow the start bit
- After the last bit of the character is transmitted, a stop bit is detected when the line returns to the 1-state for at least one bit time.

2.2.4 Asynchronous Communication Interface

An integrated circuits are available which are specially designed to provide the interface between computer and asynchronous serial communication devices, known as asynchronous communication interface or universal asynchronous receiver-transmitter(UART)

The block diagram of an asynchronous communication interface is as shown in fig 2.8 .It acts as a transmitter and also as a receiver. The interface is initialised for a particular mode of transfer by means of control byte that is loaded into its control register. The transmitter register accepts a data byte from the CPU through data bus. This byte is transferred to a shift register for serial transmission. The another shift register receives the serial information and when complete the data byte is accumulated, it is transferred to receiver register. The bits in the status register are used for input output flags and for recording certain errors that may occur during the transmission. The chip select(CS) and the read(RD) and write(WR) control lines communicates with the CPU. The chip select is used to select the interface through the address bus. The two bits in the status register are used as flags. One bit is used to indicate whether the transmission register is empty and another bit is used to indicate whether the receiver register is full.

Before using the interface following parameters must be initialized , baud rate for transmission and baud rate for reception, how many bits are in each character, whether to generate and check parity, how many stop bits are appended to each character.



CS	RS	OPERATION	REGISTER SELECTED
0	X	X	NONE-Data bus in high impedance
0	0	WR	Transmitter Register
1	1	WR	Control Register
1	0	RD	Receiver Register
1	1	RD	Status Register

Fig 2.8 UART/Asynchronous communication Interface

Transmission:

- Check the flag whether transmission register is empty
- If empty the CPU transfers a character to the transmitter register and clears the flag
- The first bit in the transmitter shift register is set to 0 to generate start bit
- The character is transferred parallel to the shift register with appended stop bits
- Then the transmitter register marked empty
- The character is transmitted bit by bit by the shift register in the transmission baud rate.
- The CPU transfer another character to the transmitter register after checking the flag

Reception:

- Once start bit is detected the incoming character bits are transmitted in to the shift register in the speed of receiver baud rate.
- After receiving the data bits the interface checks for parity and stop bits.
- The character bits without start and stop bits is then transferred in parallel from the shift register to the receiver register.
- The flag is set to indicate receiver register is full
- The CPU reads the status register and check the flag
- If set it reads the data from the receiver register

Possible error signals shown by status register: Parity error- if the number of 1's in the received data is not the correct parity, Framing Errror- the right number of stop bits is not detected at the end of the received character, Overrun Error-CPU does not read the character from the receiver register before the next one becomes available in the shift register.

2 .3 MODES OF TRANSFER

Binary information received from an external device is usually stored in memory. Information transferred from the central computer in to an external device originates from memory unit.The CPU merely execute the I/O instruction the ultimate source or destination is the memory unit.

Data transfer to and from peripherals may be handled in one of three possible modes.

- Programmed I/O
- Interrupt –initiated I/O
- Direct Memory Access

Programmed I/O operations are the result of I/O instruction written in the computer program. Each data item transfer is initiated by an instruction in the program. Interrupt initiated I/O operations are when the device is ready with data, device interrupt the CPU and transfer the data through CPU. In DMA operation the block of data are transferred between an external device and memory directly without the knowledge of CPU.

2.3.1.Programmed I/O

A transfer from am I/O device to memory requires the execution of several instructions by the CPU, An input instruction to transfer the data from the device to the CPU and a store instruction to transfer the data from the CPU to memory. The fig 2.9 shows the example of data transfer from an I/O device through an interface into the CPU.

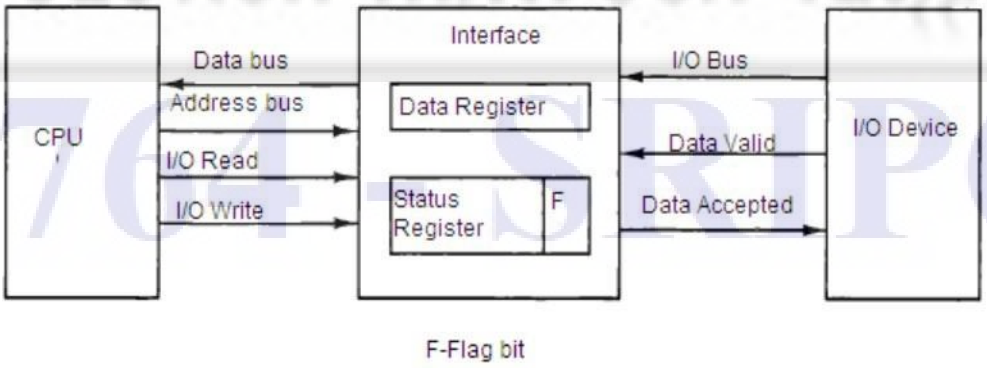


Fig 2.9 Programmed I/O transfer

The following steps are followed in the operation.

- When data is available it is placed in I/O bus and enables data valid signal

- The data register in the interface accept the data and enables data accepted signal.
- The interface set flag as1 in the status register to inform that data is ready in data register
- The data valid signal became disabled.
- The program written in the program checks the status flag if it is 0 it has to wait until it is set else if it is set the CPU reads the data from the data register.
- The flag bit becomes 0 by either CPU or the interface depends on the design
- The data accepted signal is disabled to initiate for next data transfer.

Disadvantage: The CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. i.e the CPU time is wasted for waiting for a data.

Interrupt Initiated I/O

An alternative to the CPU constantly monitoring the flag in the programmed I/O is to let the interface inform the computer when it is ready to transfer data by means of interrupt signal.

- The peripherals are provided with one or more interrupt lines on the bus.
- When a peripheral is ready with the data or ready to receive the data it raises an interrupt signal through its interface.i.e. an interrupt request is send to the CPU.
- The CPU then stop executing the current program
- Storing return address from the program counter in to a memory stack and jumps to an interrupt service routine that processes the I/O transfer
- The processor chooses the branch address of the service routine in two ways
- If the branch address(ISR) is assigned to a fixed location in memory for each device it is known as non vectored interrupt type
- If the branch address(ISR) is supplied by the device which initiate interrupt is known as vectored interrupt type.
- After the I/O transfer is done the CPU goes back to the program which it was executing before interruption..

Advantage: The CPU time is not wasted for waiting for an I/O transfer.

Vectored Interrupt:

A processing technique in which the interrupting device directing the processor to the Interrrupt Service Routine(ISR). The vectored interrupts are achieved by assigning each interrupting device a unique code typically four to eight bits in length. When a device interrupts it sends its unique code over the data bus to the processor telling the processor which interrupt service routine to execute.

Non Vectored Interrupt

A processing technique in which the user/programmer has to direct the processor to the Interrupt Service routine. When a device interrupts the processor directed to fixed memory location in the memory.

Priority Interupt

The first task of interrupt system is to identify the source of the interrupt. There is also the possibility that several sources will request service simultaneously. A priority interrupt system establishes a priority over the various sources to determine which condition is to be serviced first when two or more request arrive simultaneously. Higher priority levels are assigned to requests which are if delayed or interrupted could have serious consequences. Normally devices with high speed transfers are given higher priority and low speed transfers are given lower priority. When two devices interrupt the CPU the device with higher priority only got first preference.

There are two ways of solving the interrupt priority 1.Software 2.Hardware

Software method uses polling procedure to identify the highest priority source. i.e. check the status of each I/O module in order to know which I/O module sent the interrupt. The information can be found in the status registers of the I/O modules. Priority can be implemented on the basis of polling sequence. i.e. the highest priority source is tested first and if is interrupt signal is on control branches to a service routine for this source. Otherwise the next lower priority source is tested and so on.

Dis advantage: if there are many interrupts, the time required to poll then can exceed the time available to service the I/O device.

Hardware method accepts requests from many sources determines which of the incoming requests has the highest priority and issues an interrupt request to the CPU based on this determination. Each interrupt source has its own interrupt vector to access its own service routine directly. Thus no polling is required as the all the decisions are established by the hardware priority interrupt unit. The hardware priority unit function can be established by either a serial or a parallel connection of interrupt lines

Daisy Chaining Priority

The devices are connected in serial form. The device with the highest priority is placed in the first position followed by lower priority devices up to the device with the lowest priority which is placed last in the chain. The fig 2.10 shows the connection. The interrupt request line is common to all devices. The interrupt request signal is normally in high state when a interrupt request is initialized by any one of the device the CPU responds to it by sending Interrupt acknowledgement signal. This signal is received by the device 1 at its priority in input(PI). The acknowledgement signal passes on to the next device through Priority out output(PO) only if the device 1 is not requesting the interrupt. If the device 1 has a pending interrupt it blocks the acknowledgement signal from the next device by placing a 0in the PO output. It then proceed to insert its own interrupt vector address in to the data bus

for the CPU to use during the interrupt cycle even though it didn't send the interrupt request.

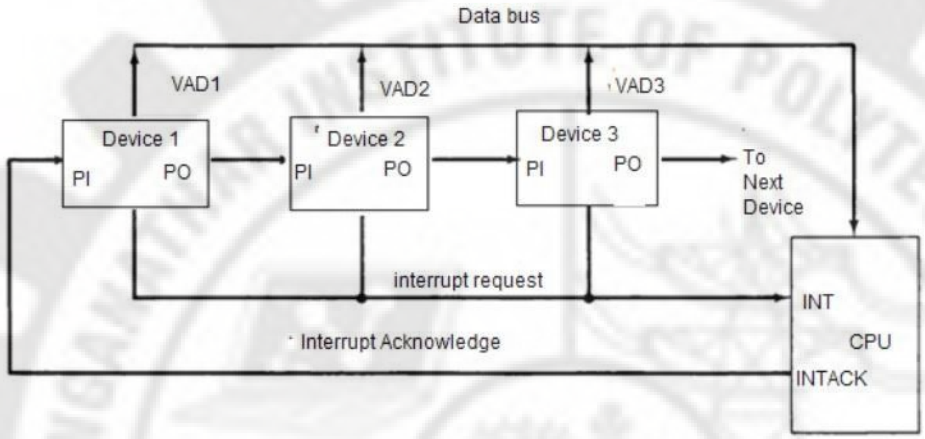


Fig 2.10 Serial Priority Interface

A device with a 0 in its PI input generates a 0 in its PO output to inform the next lower priority device that the acknowledge signal has been blocked. A device that is requesting an interrupt and has a i in its PI input will intercept the acknowledge signal by placing 0 in its PO output. If the device does not have pending interrupts is transmits the acknowledge signal to the next device by placing a 1 in tis PO output. Thus the device with PI=1 and PO=0 is the one with the highest priority that is requesting an interrupt and this device places its VAD on the data bus. The daisy chain arrangement gives the highest priority to the device that receives the interrupt acknowledge signal from the CPU.

2.3.2.5. Parallel Priority Interrupt

In this method a interrupt register is used whose bits are set separately by the interrupt signal from each device. Priority is established according to the position of the bits in the bits in the register. A mask register is used to control the status of each interrupt request. i.e. it is programmable to disable any of the priority interrupts. In a computer system interrupt operation is possible only when the interrupt enable flip flop is enabled.(IEN).

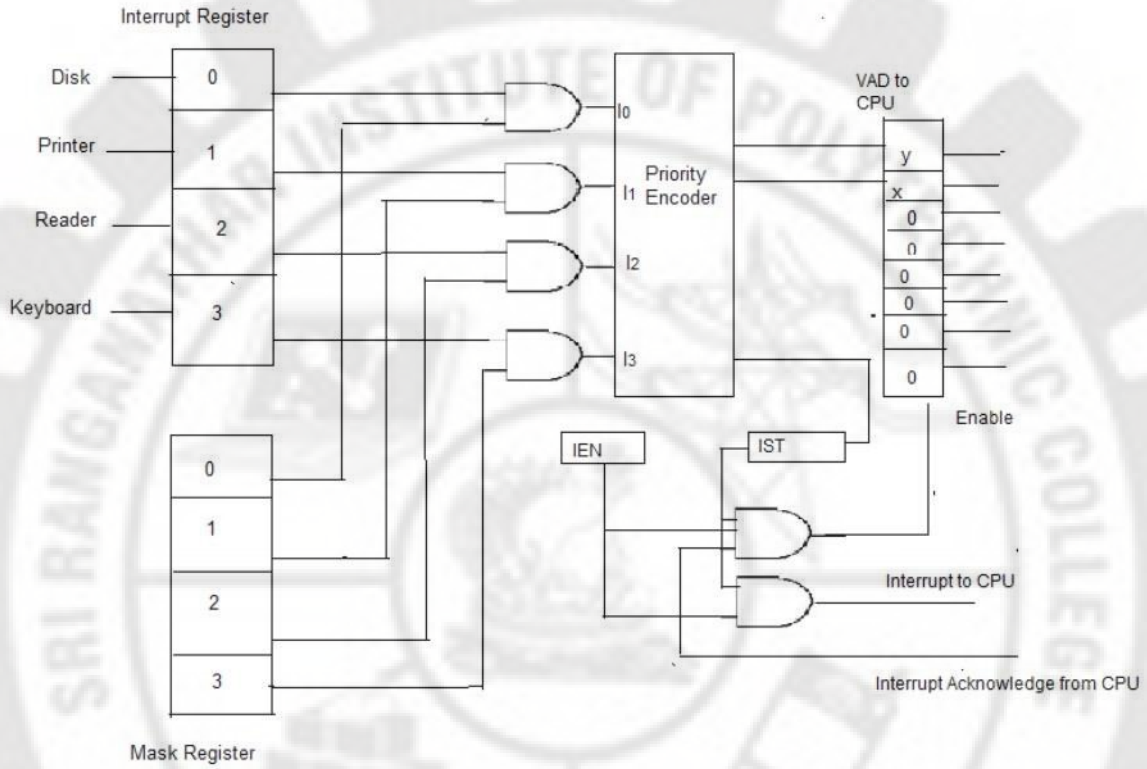


Fig 2.11 Parallel priority interrupt

Four interrupt source parallel priority interrupt circuit is shown in fig.2.11 the interrupt register whose individual bits are set by external conditions and cleared by program instructions. the devices interrupts are connected according to their transfer speed. high speed transfer device interrupt in the first position and very low speed transfer device interrupt in the last position. The mask register has the same number of bits as the interrupt register and by program instructions the bits are set or reset. the interrupt bit and corresponding mask bit are applied to an AND gate to produce the four inputs to a priority encoder. the priority encoder generates two bits of the vector address which is transferred to the CPU.

The priority encoder sets an interrupt status flip flop IST when an interrupt that is not masked occurs. The IST and IEN are AND ed and interrupt request signal is transferred to the CPU. When the interrupt acknowledge INTACK signal from the CPU enables the bus buffers in the output register and a vector address VAD is placed into the data bus.

Priority Encoder:

A circuit that implements the priority function. The logic is if two or more inputs arrive at the same time, the input having the highest priority will take precedence. The truth table of a four input priority encoder is given in table.2.1 The X 's in the table designate don't care conditions. The output of the priority encoder is used to form part of the vector address for each interrupt source. The other bits of the vector address can be assigned any value.

Table 2.1 Parity Encoder

Inputs				outputs		
I0	I1	I2	I3	X	Y	IST
1	X	X	X	0	0	1
0	1	X	X	0	1	1
0	0	1	X	1	0	1
0	0	0	1	1	1	1
0	0	0	0	X	X	0

Interrupt Controller

1. One or more of the INTERRUPT REQUEST lines (IR0 - IR7) are raised high, setting the corresponding bits in the Interrupt Request Register (IRR).

2. Then the mask register is checked for any masking of the interrupts

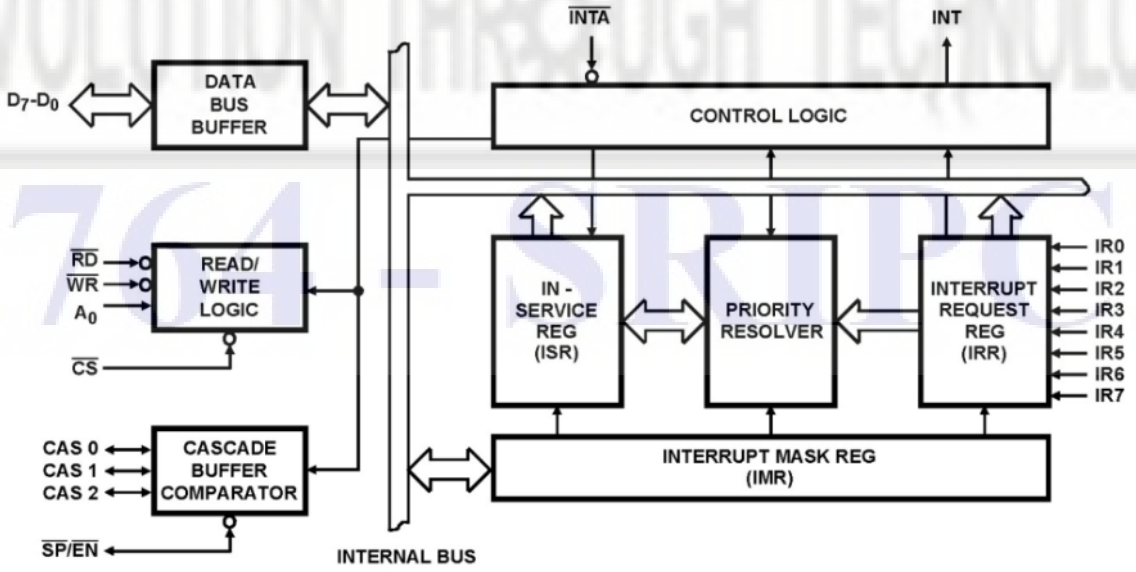
3. The interrupt is evaluated in the priority resolver. If appropriate, an interrupt is sent to the CPU via the INT line.

4. The CPU acknowledges the interrupt by sending a pulse on the INTA line. Upon reception of this pulse, the interrupt controller responds by forcing the opcode for a call instruction (0CDH) onto the data bus.

5. A second INTA pulse is sent from the CPU. At this time, the device will respond by placing the lower byte of the address of the appropriate service routine onto the data bus. This address is derived from ICW1

6. A final (third) pulse of INTA occurs, and the interrupt controller responds by placing the upper byte of the address onto the data bus. This address is taken from ICW2

7. The three byte call instruction is then complete. If the Automatic End Of Interrupt mode has been chosen, the bit set during the first INTA pulse in the ISR is reset at the end of the third INTA pulse. Otherwise, it will not get reset until an appropriate End of Interrupt command is issued to the interrupt controller.



Direct Memory Access

The transfer of data between external I/O devices and memory happens directly without the knowledge of CPU. i.e. the CPU is idle. For transferring data, buses are required from CPU. A device is required to control the transfer between I/O and memory. A DMA controller is used for DMA transfer. The fig 2.13 shows the signals in DMA transfer.

The following steps are implemented.

- The external device send DMA REQUEST for a DMA transfer in to the DMA controller
- The DMA controller request the CPU for buses using BUS REQUEST input
- The CPU response to it by giving BUS GRANT output signal
- The DMA controller Response to the external device by DMA ACKNOWLEDGE signal
- Under the supervision of DMA controller then block of data are transferred between external device and memory.

There are different types of transfer:

Burst transfer: Block of data are transferred in a continuous burst under DMA controller

Cycle stealing: allows DMA controller to transfer one data word at a time, after which it must return control to the CPU

DMA Controller

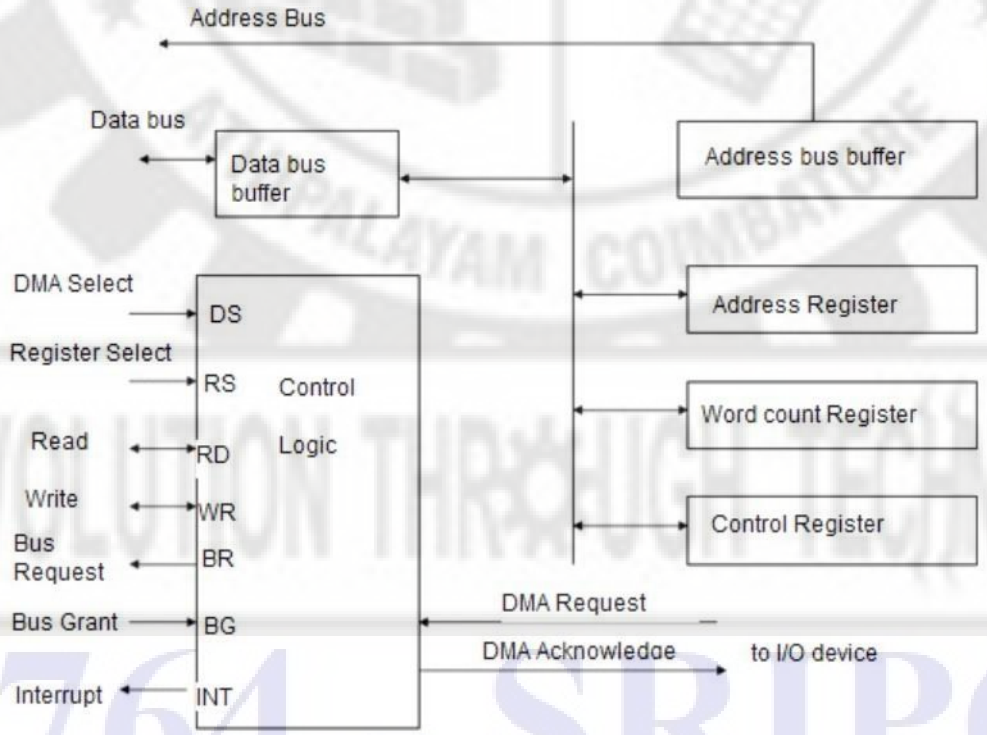


Fig 2.14 DMA Controller Block Diagram

The block diagram of the DMA controller is shown in fig 2.14.The registers present in the controller are

Address register – to hold the address location of the memory (incremented after every transfer)

Word count register – to hold the number of words to be transferred (decremented after every transfer)

Control register—specifies the mode of transfer.

Two buffer registers are present one for address bus and another for data bus. Register select signal is used for selecting the registers inside the controller for initialization.

The DMA controller needs some Initialization process before start to transfer the data between devices.

- The starting address of the memory block where data are available (for read) or where data are to be stored(for write)
- The word count which is the number of words in the memory block
- Control to specify the mode of transfer such as read or write
- A control to start the DMA transfer.DMA controller for operation.

Interrupt signal is used to inform the CPU when the block of data transfer using DMA is completed.

2.3.3.2 DMA Transfer

. The positon of the DMA controller among the other components in a computer system is shown in fig 2.15. The CPU communicates with the DMA through the address and data buses as with any interface unit. The DMA has its own address which activates the DS and RS lines. The CPU initialize the DMA through data bus. Once the DMA receives the start control command it can start the transfer between peripheral and memory

- Peripheral device sends DMA Request line to DMA controller
- DMA controller activates BR line
- The CPU responds with BG line
- The DMA puts the current value of its address register into the address bus, initiates RD and Wr signal and send DMA acknowledge line to the peripheral
- The RD and WR lines in the DMA controller are bidirectional. The direction of transfer depends on the status of the BG line.
- When BG=0 the RD and WR are input lines allowing the CPU to communicate with the internal DMA registers.
- When BG =1 the RD and WR are output lines from the DMA controller to the RAM to specify the read or write operation for the data.
- If the word count register reaches zero, the DMA stops any further transfer and removes its bus request.
- It also informs the CPU of the termination by means of an interrupt.

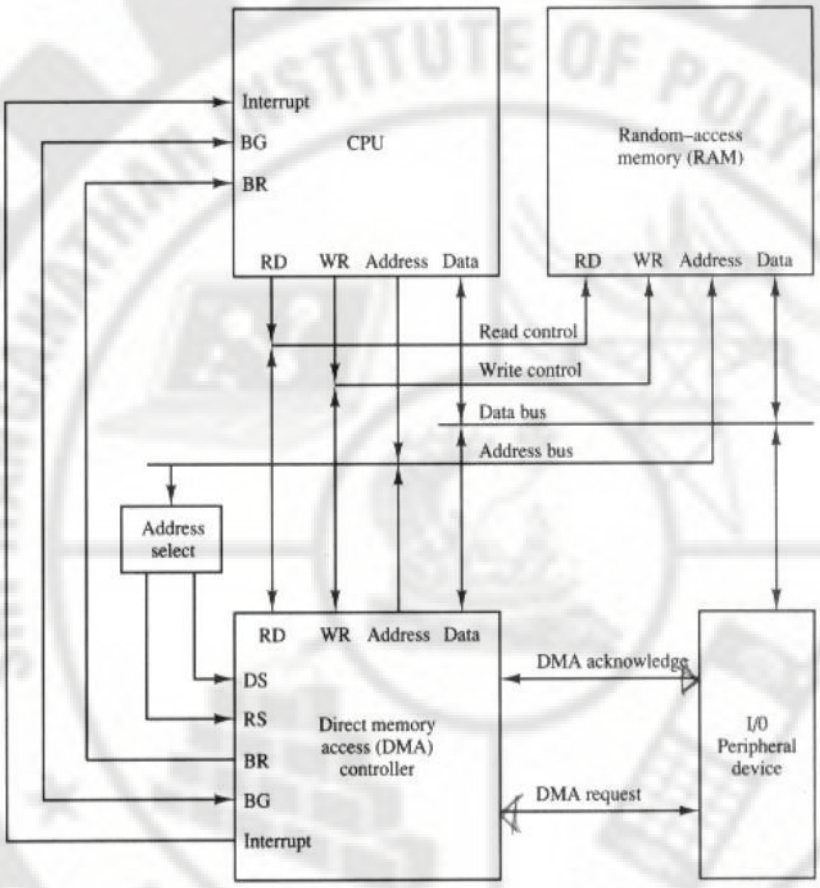


Fig 2.15 DMA transfer in a computer system.

2.4.INPUT OUTPUT PROCESSOR(IOP)

An input –output processor may be classified as a processor with direct memory access capability that communicates with I/O devices. A processor that communicates with remote terminals over telephone and other communication media in a serial mode is called a data communication processor.

The memory unit occupies a central position and communicate with each processor by means of DMA. The responsible for processing data needed in the solution of computational tasks. The IOP provides a path for transfer of data between various peripheral devices and the memory unit. The CPU is usually assigned the task of initiating the I/O program. From then on the IOP operates independent of the CPU and continues to transfer data from external devices and memory.

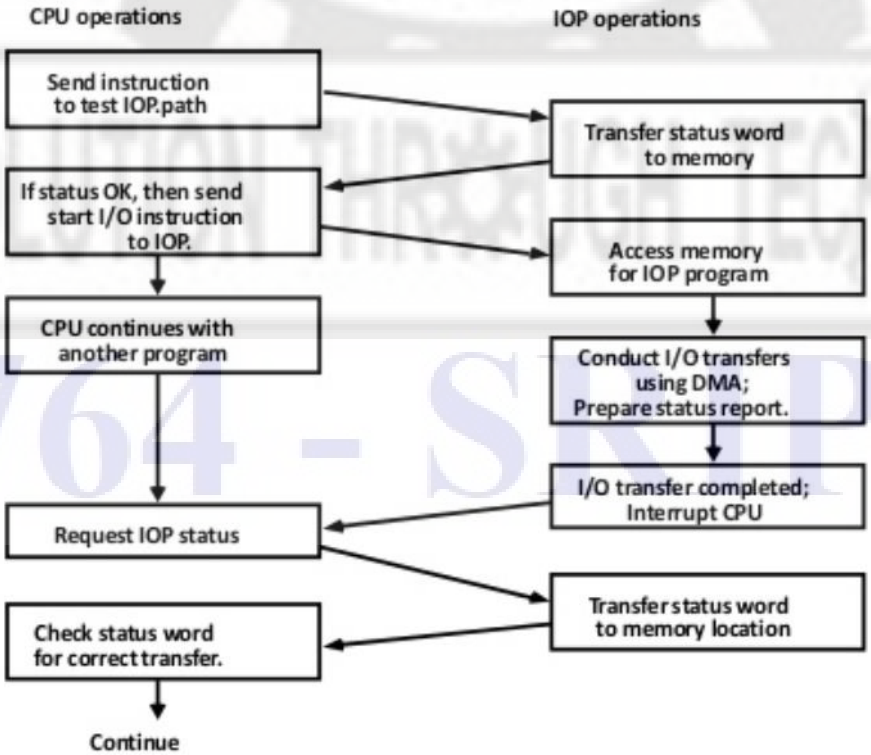
The CPU is the master and IOP is a slave processor. The CPU is assigned the task of initiating all operations, but I/O instructions are executed in the IOP. CPU instructions provide operations to start an I/O transfer and also to test I/O status conditions needed for making decisions on various I/O activities. The IOP by means of interrupt attract CPU attention. It also responds to CPU requests by placing a status word in a prescribed location in memory to be examined later by a CPU program. When I/O operation is desired the CPU informs the IOP where to find the I/O program and then leaves details to the IOP.

CPU-IOP Communication

The communication between CPU and IOP may take different forms depending on the particular computer considered.

The sequence of operations may be carried out are given below.

- The CPU sends a test I/O instruction to IOP to test the IOP path.
- The responds by inserting a status word in memory location.
- The CPU refers to the status word in memory. If everything is in order, the CPU sends the start I/O instruction to start the I/O transfer.
- The IOP accesses memory for IOP program.
- The CPU can now continue with another program while the IOP is busy with the program. Both programs refer to memory by means of DMA transfer.
- When the IOP terminates the execution of its program, it sends an interrupt request to the CPU.
- The CPU then issues a read I/O instruction to read the status from the IOP.
- The IOP transfers the status word to memory location.
- The status word indicates whether the transfer has been completed satisfactorily or if any error has occurred during the transfer.



Serial Communication:

A specialised I/O processor designed to communicate directly with data communication networks for serial transfer. A data communication processor communicates with each terminal through a single pair of wires. Both data and control information are transferred in a serial fashion with the result that the transfer rate is much slower. The task of the data communication processor is to transmit and collect digital information to and from each terminal, determine if the information is data or control and responds to all requests according to predetermined established procedures. The processor obviously must also communicate with the CPU and memory in the same manner as any I/O processor.

Since telephone lines were originally designed for voice communication and computers communicate in terms of digital signals some form of converters must be used. A modem converts digit signal into audio tones to be transmitted over telephone lines and also converts audio tone from the line to digital signals for machine use. Various modulation schemes as well as different grades of communication media and transmission speeds are used. A communication line may be connected to a synchronous or asynchronous interface depending on the transmission method of the remote terminal.

Different modes of data transfer are

A simplex transmission - line carries information in one direction only

A half duplex transmission -can send and receive data in both directions but data can be transmitted in only one direction at a time.

A full duplex transmission- can send and receive data in both directions simultaneously.

Summary:

- Input and output devices are connected to a CPU through Interface.The main function of input output interface circuit should be data conversion, synchronization and device selection
- Communication between two devices may be synchronous or asynchronous.In asynchronous transfer clock signal is different for source and destination.Asynchronous transfer taken place in two methods—strobe, handshaking
- Strobe-single line control, source initiated and destination initiated, disadvantage is source does not know whether destination received the data or not and destination does not know whether source placed the data or not
- Handshaking – two line control, source initiated and destination initiated
- The transfer of data between memory and I/O devices are in different modes.The modes are programmed I/O, Interrupt initiated, and DMA transfer

a. Programmed I/O—transfer the data when the program instruct it-Disadvantage is CPU has to wait till the device has to be ready for transfer

b. Interrupt Initiated I/O---when the device is ready for transfer it inform the CPU by means of Interrupt signal. The interrupt is acknowledged by the CPU and transfer takes place.In the interrupt method, if two or more interrupt simultaneously arrived, which interrupt has to be serviced by the CPU has to be decided by hardware in two methods. Serial method, parallel method

- Serial method is known as daisy chain priority-always higher priority interrupt are serviced even though lower priority interrupt first request the CPU and the higher priority request the CPU later.
- Parallel method interrupt is masked by software instruction, a priority encoder, interrupt enable flip flop are to solve the priority interrupt.Interrupt controller is used for this type of transfer, the CPU time is not wasted for waiting for I/O device

c. DMA transfer-bulk of data transfer use this, the transfer taken place without CPU knowledge.By means of bus request signal, bus control are taken from CPU and after transfer the bus grant signal , bus control are given back to CPU.

- DMA controller is used for this type of transfer (i.e.) control the bus and transfer the data instead of CPU.The transfer is of two types-burst mode , cycle stealing mode.Burst mode-get the bus from CPU, transfer bulk of data then give back the bus to CPU.Cycle stealing mode- get bus for single cycle from CPU and transfer only one data

- IOP is used to communicate with CPU for i/o operation.Serial communication methods-simplex, half duplex and full duplex

Review Questions

Two marks

1. Define I/O interface
2. Write the functions of I/O interface
3. Mention the usage of Device controller
4. What is the use of Port in I/O interface
5. What are the I/O commands
6. Define synchronous communication
7. Define asynchronous communication
8. Name the single line controlled data transfer method
9. Name the double line controlled data transfer method
10. Mention the types of 8 bit data transfer between two units
11. asynchronous data transfer why we need start and stop bits
12. Name some points for moving towards handshaking method from strobe method.
13. Name the signals used in handshaking method'
14. What are the different modes of data transfer
15. What is the advantage of interrupt initiated data transfer
16. Define vectored interrupt

17. Define non vectored interrupt
18. What is Interrupt service routine
19. When we are choosing DMA data transfer
20. What is block transfer
21. Why the name cycle stealing arise
22. Mention the disadvantage of using programmed I/O
23. A set in the status flag of the I/O interface implies what?
24. Name the software method of resolving priority interrupt
25. How the interrupt initiated i/o transfer occurs
26. Define DMA
27. What are the different modes of DMA transfer
28. What is serial communication
29. What are the types of serial transfer

Three marks

1. Draw a circuit for connecting a DVD and CD drive to a CPU
2. What are the steps taken place while doing source initiated storbe method of data transfer
3. What are the steps taken place while doing destinatopm initiated storbe method of data transfer
4. What are the steps taken place while doing source initiated handshaking method of data transfer
5. What are the steps taken place while doing destination initiated handshaking method of data transfer
6. To transfer 1010110010 bit data in asynchronous mode draw the bit format
7. Mention the transmitter function of Asynchronous communication interface
8. Mention the Receiver function of Asynchronous communication interface
9. What are the initial controls to be made for using asynchronous communication interface
10. Mention the function CPU when transferring data to a peripheral in programmed i/o mode
11. What are the steps taken place while transferring data in interrupt initiated mode
12. Mention the registers present in interrupt controller
13. Why the software polling method is not selected for solving interrupt priority
14. What is the use of IEN in parallel priority resolver and when the IST is set
15. Draw a priority encoder circuit with its truth table
16. Mention the signals available in DMA data transfer
17. In DMA transfer what is the major role of BG signal
18. What is IOP
19. Write a short note about modem

Five marks

1. Explain strobe controlled data transfer
2. Explain source initiated handshaking method of data transfer
3. Explain destination initiated handshaking method of data transfer
4. Explain daisy chain priority
5. Explain parallel priority interrupt
6. Explain interrupt controller operation
7. Explain DMA controller operation
8. Explain Programmed I/O transfer
9. Explain the operation of interrupt initiated i/o transfer operation
10. Explain the DMA transfer
11. Draw the flow chart explaining CPU-IOP communication

UNIT III. MEMORY MODULES

OBJECTIVES

- To Know different types of storage devices available in a system
- To understand which types of semiconductor memory constitute main memory.To design a main memory using RAM and ROM
- To Know about different secondary storage devices
- To know about cache memory concept, the initialization, the different mapping techniques, and how to write data into cache
- To understand the concept of virtual memory management and its mapping techniques and to know about different page replacement techniques.

INTRODUCTION

The computer's programs and the data they operate on are held in some place of the computer during execution. The place is commonly known as Memory. we describe the different types of memory available in a computer system. know about the most common components and organizations used to implement the main memory. the apparent speed of the main memory can be increased by means of caches. we present the virtual memory concept which increases the apparent size of the memory.

MEMORY TYPES

The memory of a computer is not actually concentrated in one place. Storage devices are scattered throughout the machine

CPU Registers

A register is a group of flip flops with each flip flop capable of storing one bit information. A n -bit register has a group of n flip flops and is capable of storing any binary information of n bits. They are all present in the CPU and called as CPU registers. The arithmetic and control sections of the computer use these registers for their operation. (i.e.) temporarily store the information. Arithmetic operations, including additions, multiplications shifts etc are all performed in these registers of the machine.

Main Memory

The next category of storage device encountered is called the high speed memory, inner memory, main memory. The main memory is the central storage unit of a computer system. It is relatively large and fast memory to store programs and data during computer operation. Semiconductor integrated circuits are used for the main memory. RAM and ROM are the two main memory types.

ROM is a random access memory with only read operation, storing programs that are permanently resident in the computer and for table of constants that do not change in value once the production of the computer is completed. The ROM is needed for storing bootstrap

loader program whose function is to start the computer software computing when the power is on. The contents of ROM remain unchanged after power is turned off or on. The bootstrap program loads a portion of the operating system from disk to main memory and control is then transferred to the operating system which prepares the computer for general use.

RAM is a random access memory with read and write operation storing programs that are temporarily stored in the computer for processing. There are two types of RAM Static and dynamic. Dynamic RAMs are using capacitor for storage. The stored charges on the capacitor are tend to discharge with time and it must be periodically recharged. The dynamic RAM offers reduced power consumption and larger storage capacity in a single memory chip. So they are used as main memory.

Thus DRAM and ROM are playing as main memory for a computer system.

Secondary Memory

The presently available devices that are fast enough to perform function satisfactorily do not possess the storage capacity that is sometimes required. Additional memory called auxiliary memory or secondary memory is added in most computers. The secondary memory is low cost per digit stored but operating speed is less compared to register and main memory.

Cache Memory

The next category of storage device encountered is called the high speed memory than main memory, the cache memory. SRAM is playing as cache memory for a computer system. Static RAMs are using FLIP FLOPs for storage and keep the data until the power is on. Static RAM is easier to use and has shorter read and write cycles.

MAIN MEMORY

RAM and ROM are available in different sizes. The number of chips are combined according to the users requirement.

3.2.1Read Only Memory

A memory unit that randomly accessing and performs the read operation only is known as ROM. The binary information stored in a ROM is made permanent during the hardware production of the computer and cannot be altered by writing different words into it. ROMs come with special internal electronic fuses that can be “programmed “ for a specific configuration . Once the pattern is established it stays within the unit even power is turned off and on again. A m x n ROM is an array of binary cells organized into m words of n bits.It has k address input lines to select = m words of memory and n output lines. A ROM IC have one or more enable inputs for expanding a number of packages into a ROM with larger capacity.ROM does not need a read control line since at any given time the output lines automatically provide the n bits of the word selected by the address value.

Example: The fig 3.1 shows a ROM chip of m=512 ,n=8 . To find the number of address lines required =m is the formula. =512 . So, nine address lines in the ROM chip specify any one of the 512 bytes stored in it. The two chip select inputs must be CS1=1 and CS2=0 for the unit to operate.

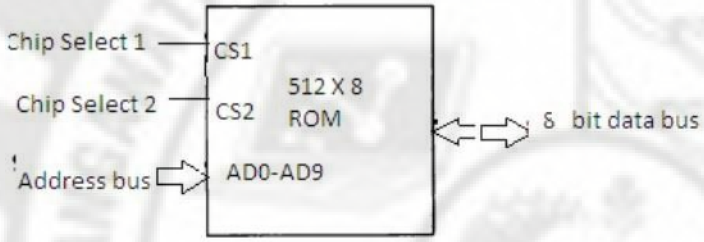


Fig 3.1 ROM Chip

3.3. Random Access Memory

The process of locating a word in memory is the same and requires an equal amount of time no matter where the cells are located physically in memory is known as random access memory. Communication between a memory and its environment is achieved through data input and output lines, address selection lines, and control lines that specify the direction of transfer.

The n data input lines provide information to be stored in memory and n output data lines supply the information from the memory. The k address lines provide a binary number of k bits that specify a particular word chosen among the available inside the memory.The two control inputs specify the direction of transfer desired.

The two operations that a random access memory can perform are the read and write operations. The steps that must be taken for the purpose of transferring a new word to be stored into the memory are as follows.

- Apply the binary address of the desired word into the address lines
- Apply the data bits that must be stored in memory into the data input lines
- Activate the write input
- The memory unit will then take the bits presently available in the input data lines and store them in the word specified by the address lines

The steps that must be taken for the purpose of transferring a stored word out of memory are as follows

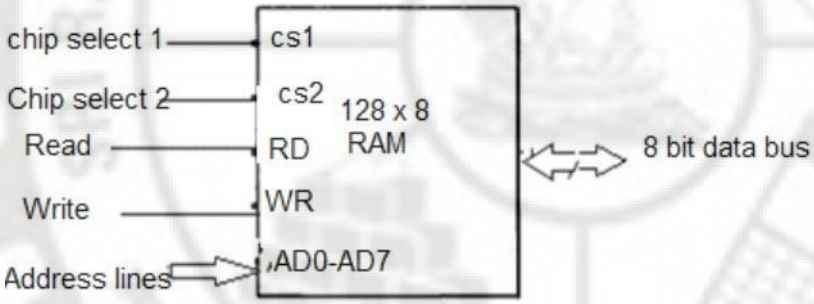
- Apply the binary address of the desired word into the address lines
- Activate the read input
- The memory unit will then take the bits from the word that has been selected by the address and apply them into the output data lines.

- The content of the selected word does not change after reading

Example:

The RAM has bidirectional data lines with tri state buffers. The buffer outputs signal for 0, signal for 1 and a high impedance state. The high impedance state behaves like an open circuit which means that the output does not carry a signal and has no logic significance.

. The fig 3.2 shows a RAM chip of m=128 ,n=8 . To find the number of address lines required =m is the formula. =128. So, seven address lines in the ROM chip specify any one of the 218bytes stored in it. The two chip select inputs must be CS1=1 and CS2=0 for the unit to operate.



Cs1		RD	WR	MEMORY FUNCTION	STATE OF DATA BUS
0	0	X	X	Inhibit	High-Impedance
0	1	x	X	Inhibit	High-Impedance
1	0	0	0	Inhibit	High-Impedance
1	0	0	1	Write	Input data to RAM
1	0	1	X	Read	Output data from RAM
1	1	x	X	Inhibit	High-Impedance

. The read and write inputs specify the memory operation and the two chip select control inputs are for enabling the chip only when it is selected by the microprocessor. The read and write inputs are sometimes combined into one line labeled R/.

3.2.3 Memory Address Map

The designer of the computer system must calculate the amount of memory required for the particular application and assign it to either RAM or ROM. The interconnection between memory and processor is established from the knowledge of the size of the memory needed and type of RAM and ROM chips available. The table called memory address map is a pictorial representation of assigned address space for each chip in the system.

Table 3.1 memory address map

COMPONENT	HEXA DECIMAL ADDRESS	ADDRESS BUS LINES									
		10	9	8	7	6	5	4	3	2	1

RAM1	0000	-	007F	0	0	0	X	X	X	X	X	X	X
RAM2	0080	-	00FF	0	0	1	X	X	X	X	X	X	X
RAM3	0100	-	017F	0	1	0	X	X	X	X	X	X	X
RAM4	0180	-	01FF	0	1	1	X	X	X	X	X	X	X
ROM	0200	-	03FF	1	X	X	X	X	X	X	X	X	X

the requirement be 512 bytes RAM and 512 bytes ROM. Available chips are 512 bytes ROM and 128 bytes RAM. Therefore the solution is using one 512 bytes ROM chip and four 128 bytes RAM chip for the requirement. The memory address map for this configuration is shown in table 3.1.

The component specifies RAM/ROM.The hexadecimal address column assigns a range of hexadecimal equivalent addresses for each chip. The address bus lines are listed in the third column. The table shows 10 lines of address the remaining higher address lines are "0". For the RAM chips 7 address lines are used for addressing and 2 address lines are used for selecting the RAM chip for operation. For the ROM chip 9 address lines are used for addressing.one line(i.e) 10^{th} address line is used for selecting RAM /ROM.

3.2.4.Memory Connection to CPU

RAM and ROM chips are connected to a CPU through the data and address buses. Each RAM receives the seven low order bits of the address bus to select one of the 128 bytes . The particular RAM chip selected is determined from line 8 and 9 in the address bus. This is done through 2 x 4 decoder whose outputs go to the CS1 inputs in each RAM chip. The RD and WR outputs from the microprocessor are applied to the inputs of each RAM chip.

The selection between RAM and ROM is achieved through bus line 10. The RAMs are selected when the bit in this line is 0 and the ROM when the bit is 1. The other chip select input in the ROM is connected to the RD control line of the ROM chip to be enabled only during a read operation. The data bus of the ROM has only an output capability whereas the data bus connected to the RAMs can transfer information in both directions.

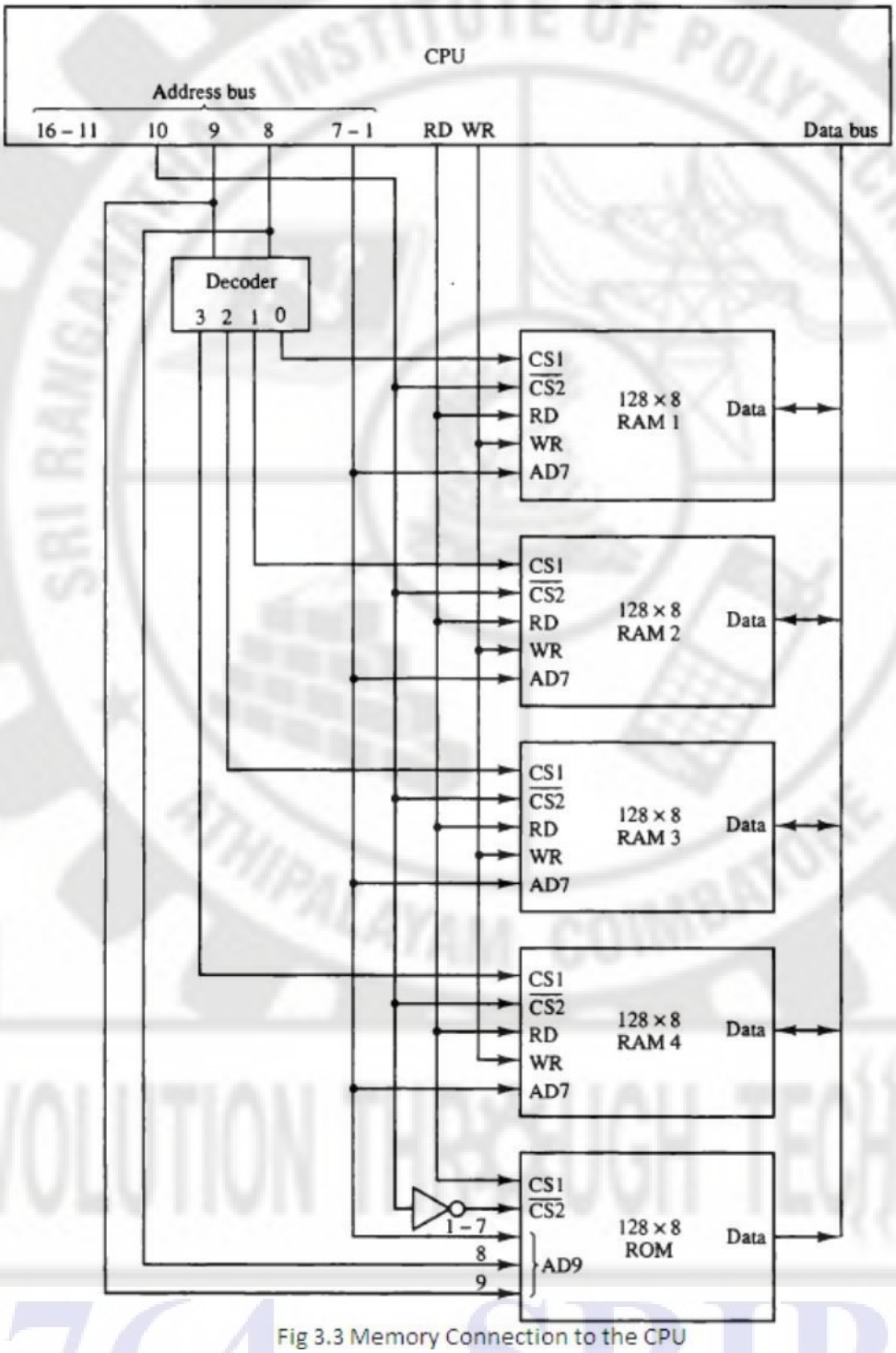


Fig 3.3 Memory Connection to the CPU

SECONDARY MEMORY

Magnetic Tape

A magnetic tape transport consists of the electrical, mechanical and electronic components to provide the parts and control mechanism for a magnetic tape unit. The tape itself is a strip of plastic coated with magnetic recording medium. Bits are recorded as magnetic spots on the tape along several tracks. Usually seven or nine bits are recorded simultaneously to form a character together with a parity bit. Read/write heads are mounted one in each track so that data can be recorded and read as a sequence of characters.

Magnetic tape units can be stopped, started to move forward or in reverse or can be rewound. However they cannot be started or stopped fast enough between individual characters. For this reason information is recorded in blocks referred to as records. Gaps of unrecorded tape are inserted between records where the tape can be stopped. The tape starts moving while in a gap and attains its constant speed by the time it reaches the next record. Each record on tape has an identification bit pattern at the beginning and end. By reading the bit pattern at the beginning the tape control identifies the record number. By reading the bit pattern at the end of the record the control recognizes the beginning of the gap. A tape unit is addressed by specifying the record number and the number of characters in the record. Records may be of fixed or variable length.

Disadvantage: sequential access and access time depends on the position of record in the tape.

Magnetic disk:

A disk is a circular plate constructed of metal or plastic coated with magnetized material. Often both sides of the disk are used and several disks may be stacked on one spindle with read/write heads available on each surface. All disks rotate together at high speed and are not stopped or started for access purposes. Bits are stored in the magnetized surface in spots along concentric circles called tracks. The tracks are commonly divided into sections called sectors. Some units use a single read/write head for each disk surface. In this type of unit the track address bits are used by a mechanical assembly to move the head into the specified track position before reading or writing. In other disk systems separate read/write heads are provided for each track in each surface. The address bits can then select a particular track electronically through a decoder circuit.

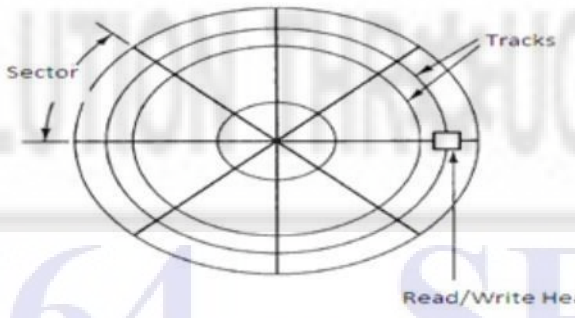


Fig 3.4 Magnetic Disk

Permanent timing tracks are used in disks to synchronize the bits and recognize the sectors. A disk system is addressed by address bits that specify the disk number , disk surface , the sector number and the track within the sector. After the read/write heads are positioned in the specific track the system has to wait until the rotating disk reaches the specified sector under the read/write head. Information transfer is very fast once the

beginning of a sector has been reached. Disks may have multiple heads and simultaneous transfer of bits from several tracks at the same time.

A track in a given sector near the circumference is longer than a track near the center of the disk. If bits are recoded with equal density some tracks will contain more recorded bits than others. To make all the records in a sector of equal length some disks use a variable recording density with higher density on tracks near the center than on the tracks near the circumference, this equalizes the number of bits on all tracks of a given sector.

Disks that are permanently attached to the unit assembly and cannot be removed by the occasional user are called hard disks.

CACHE MEMORY

If the active potions of the program and data are place in a fast small memory the average access time can be reduced thus reducing the total execution time of the program. Such fast memory is referred to as cache memory. The cache memory access time is less than the access time of main memory by a factor of 5 to 10. It is the fastest component and approaches the speed of CPU components.

Cache memory is also often called CPU memory and it is usually physically located on the CPU. The data that is stored in cache is usually the data and commands/instructions most often used by the CPU. It is a very fast way to serve data to the processor, but the size of memory cache is limited.

Most modern CPUs have three different types of cache memory. The first, called L1 cache, is the quickest and is the first place that a CPU will look when it needs data. However, it is also the smallest of the three types of cache memory. The second type of cache — and the second place that a CPU looks for data — is called L2 cache. It is slightly slower than L1 cache, but is slightly bigger so it holds more information.

The final type of cache memory is call L3 cache. It is the third place that the CPU uses before it goes to the computer's main memory. L3 cache is the biggest cache and, despite being the slowest of the three, is still quicker than main memory.

3.4.1 Need for Cache

The study of complex programs has revealed that, at a particular point of time, references to memory are limited to a few localized portions in memory, this phenomenon is referred to as the property of "*locality of reference*".

Example for locality of reference: whenever a loop is executed in a program , the CPU refers to the group of instructions in memory repeatedly. i.e. same subroutine is called repeatedly. Thus, loops and subroutines tend to localize the references to the memory for getting instructions. To some extent, memory references to data also tend to be localized. In table look up procedures, only that portion of the memory is repeatedly referred which stores the table. Iterative procedures refer to the common memory locations and array of numbers are confined within a local portion of the memory. The result of these observations is to high light

the locality of reference property , which states that only a few localized areas of memory are referred repeatedly, while the remaining areas of memory are accessed infrequently . The fundamental idea of cache organization is that by keeping the most frequently accessed instructions and data (i.e) locality of reference in the fast cache memory, the average access time will approach the access time of the CPU.

The table 3.2 also shows the cost ,speed, and typical capacities for three popular memory technologies. From the table, the access time of the cache memory is very small and the capacity is minimum. To equate average access time of the CPU with the access time of the cache memory, the locality of references are kept in cache.

Technology	**Typical Access time**	**Economical size**	**Approximate relative cost per byte**
StaticRAM(for cache)	10ns	1MB	X
DynamicRAM(for main memory)	50ns	1GB	x/10
Disk	10ms	100GB	x/1000

Principle of Operation.

The basic operation of the cache is when the CPU needs to access instruction or data , the cache is examined if the word is found it is read from the fast memory. If the word addressed by the CPU is not found in the cache the main memory is accessed to read the word. A block of words containing the one just accessed is then transferred from main memory to cache memory. The block size may vary from one word to about 16 words adjacent to one just accessed.

The performance of cache memory is frequently measured in terms of a quantity called hit ratio. When the CPU refers to memory and finds the word in cache it is said to produce hit. If the word is not found in cache it is in main memory and it count as miss. The ratio of the number of bits divided by the total CPU references to memory (hit+miss) is the hit ratio. For best performance the hit ratio should be high.

Cache Initialization

The cache is initialized when power is applied to the computer or when the main memory is loaded with a complete set of programs from auxiliary memory. After initialization the cache is considered to be empty, but in effect it contains some non valid data. It is customary to include with each word in cache a valid bit to indicate whether or not the word contains valid data.

The cache is initialized by clearing all the valid bits to 0. The value bit of a particular cache word is set to 1 the first time this word is loaded from main memory and stays set unless the cache has to be initialized again. The introduction of valid bit means that a word

in cache is not replaced by another word unless the valid bit is set to 1 and a mismatch of tags occurs. If the valid bit happens to be 0 the new word automatically replaces the valid data. Thus the initialization condition has the effect of forcing misses from the cache until it fills with valid data.

Different Mapping Techniques

The basic characteristic of cache memory is its fast access time. So very little or no time may be wasted when searching for words in the cache. The transformation of data from main memory to cache memory is referred to as a mapping process. Three types of maaping procedures are in cache

- Associative mapping
- Direct mapping
- Set Associative mapping

Assoicative mapping:

The fastest and most flexible cache organization uses an associative memory. The associative memory stores both the address and content (data) of the memory word. This permits any location in cache to store any word from main memory. The fig 3.5 shows the associative mapping.

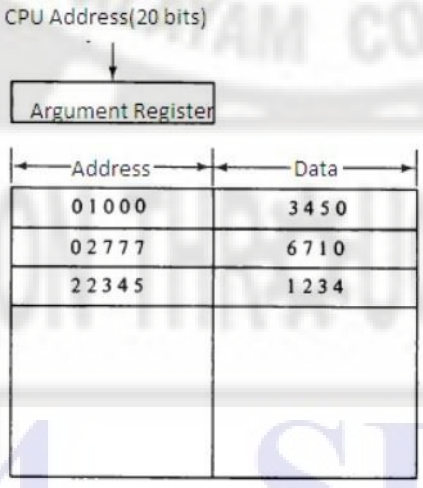


Fig 3.5 Associative Mapping Cache

The address is 20 bits and data is 16 bit. A CPU address of 20 bits is placed in the argument register and the associative memory is searched for a matching address .If the address is found the corresponding 16 bit data is read and sent to the CPU.

If no match occurs the main memory is accessed for the word. The address-data pair is then transferred to the associative cache memory from main memory..

If the cache is full and no place for new data, an address –data pair must be displaced to make room for a pair that is needed. The replacement algorithm choose by the designer determine the pair to be replaced. A simple procedure is to replace cells of the cache in round robin order whenever a new word is requested from main memory. i.e. first in first out policy.

Disadvantage: *Associative memories are expensive compared to random access memories because of the added logic associated with each cell.*

Direct Mapping

In general case there are words in cache memory and words in main memory. The n-bit memory address is divided into two fields. K-bits for the index field and n-k bits for the tag field.The direct mapping cache organization uses the n-bit address to access the main memory and the k-bit index to access the cache.

For example , the main memory size is one mega byte and the cache memory size is four kilo byte, then CPU address of 20 bits is divided into two fields. The twelve least significant bits constitute the index field and the remaining eight bits form the tag field. The number of bits in the index field is equal to the number of address bits required to access the cache memory.

The internal organization of the words in the cache memory is shown in fig 3.6

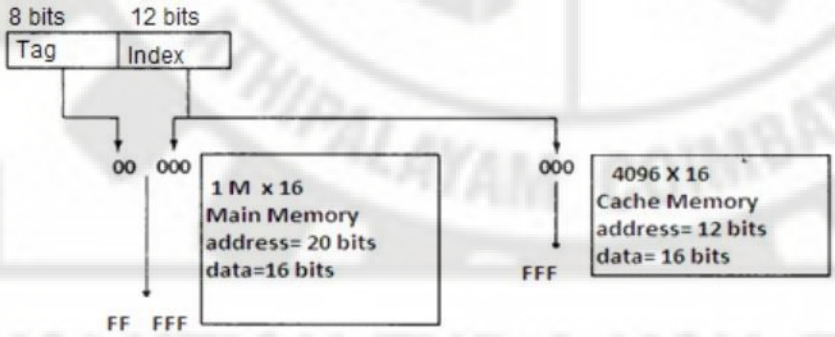


Fig 3.6 Relation between main memory address and Cache memory address

Each word in cache, consists of the data word and its associated tag. When a new word is first brought into the cache, the tag bits are stored alongside the data bits. When the CPU generates a memory request, the index field is used for the address to access the cache. The tag field of the CPU address is compared with the tag in the word read from the cache. If the two tags match there is a hit and the desired data word is in cache. If there is no match, there is a miss and the required word is read from main memory. It is then stored in the cache together with the new tag, replacing the previous value

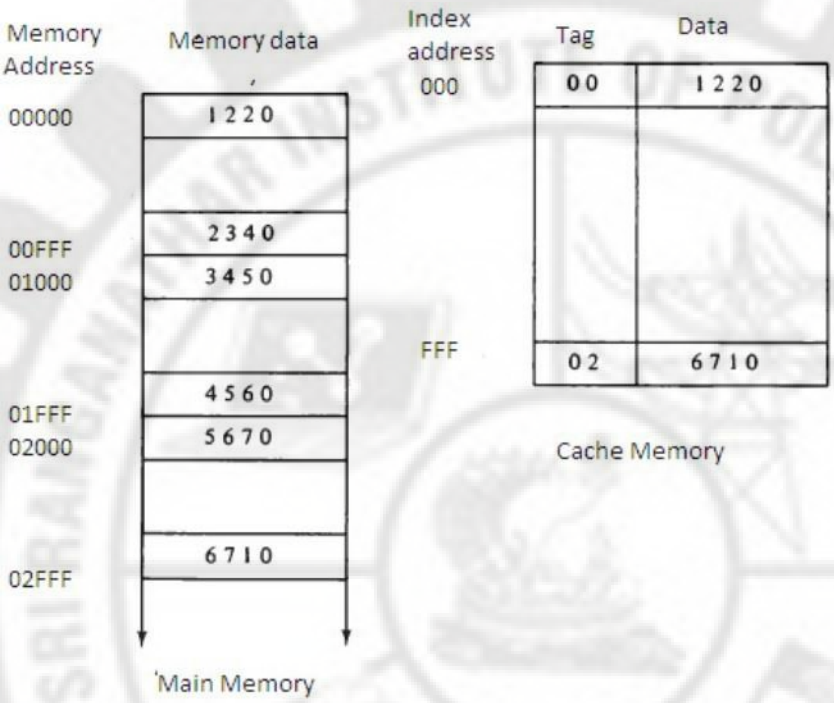


Fig 3.7 Direct Mapping Cache Organization

Disadvantage: the hit ratio can drop considerably if two or more word whose addresses have the same index but different tags are accessed repeatedly.

The direct mapping using a block size of 16 words is shown in fig 3.8. The index field is now divided into two parts. The block field and the word field. In a 16k word cache there are 256 blocks of 16 words each, since 256 x 16= 4K.The block number is specified with a 8 bit field and word within the block is specified with a 4 bit field. The tag field stored within the cache is common to all four words of the same block. Everytime a miss occurs, an entire block of four words must be transferred from main memory to cache memory.

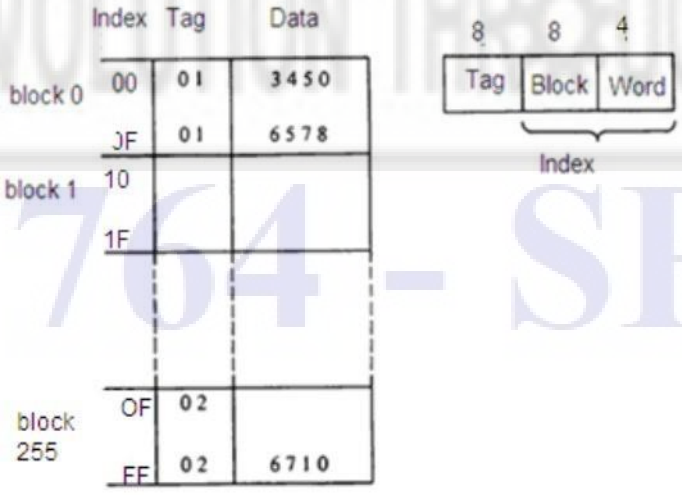


Fig 3.8 Direct mapping cache with block size of 4 words

Advantage: The hit ratio improve with a larger block size because of the sequential nature of computer programs.

Set Associative Mapping

The set associative mapping is an improvement over the direct mapping organization in that each word of cache can store two or more words of memory under the same index address. Each data word is stored together with its tag and the number of tag-data items in one word of cache is said to form a set. In general a set associative cache of set size k will accommodate k words of main memory in each word of cache.

An example of a set associative cache organization for a set size of two is shown in fig 3.9. Each index address refers to two data words and their associated tags. Each tag requires 8 bits and each data word has 16 bits, so the word length id 2(8+16)= 48 bits. An index address of 12 bits can accommodate 4096 words. Thus the size of cache memory is 4096 x 48.

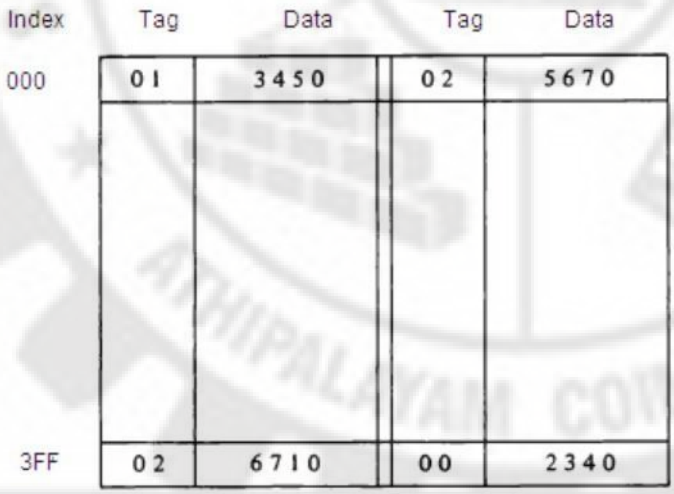


Fig 3.9 Two-way set-associative mapping Cache

When the CPU generates a memory request, the index value of the address is used to access the cache. the tag field of the CPU address is then compared with both tags in the cache to determine if a match occurs. The comparison logic is done by an associative search of the tags in the set similar to an associative memory search. thus the name “set-associative”. The hit ratio will improve as the set size increases because more words with the same index but different tags can reside in cache.

3.4.5 Writing into Cache

The simplest and most commonly used procedure is to update main memory with every write operation, with cache memory being updated in parallel if it contains the word at the specified address. This is called *WRITE THROUGH* method. This method has the advantage that ,an memory always contains the same data as the cache. This characterstic is important in system with DMA transfers. It ensures that the data residing in main memory

are valid at all times so that an I/O device communicating through DMA would receive the most recent updated data.

The second procedure is called the *WRITE BACK* method. The cache location is only updated during the write operation. The location is then marked by a flag so that later when the word is removed from the cache it is copied in to the main memory. The reason for the write back method is that during the time a word resides in the cache it may be updated several times. However as long as the word remains in the cache, It does not matter whether the copy in main memory is out of date, since request from the word are filled from the cache. It is only when the word is displaced from the cache that an accurate copy need to be rewritten into main memory. The disadvantage is DMA type of data transfer is not transferring the current data.

MEMORY MANAGEMENT

In most computer systems, the physical main memory is not as large as the address space spanned by an address issued by the processor. When a program does not completely fit into the main memory the program is placed in secondary memory and all parts of a program that are eventually executed are first brought into the main memory. when a new segment of a program is to be moved into a full memory it must replace another segment already in the memory. In modern computers a memory management is necessary to look after the process. The operating system moves programs and data automatically between the main memory and secondary storage. Thus doing memory management.

3.5.1. Virtual Memory Concept

Virtual memory is a concept used in some large computer systems that permit the user to construct programs as though a large memory space were available, equal to the totality of auxiliary memory. Each address that is referenced by the CPU goes through an address mapping from the so-called virtual address to a physical address in main memory. Virtual memory is used to give programmers the illusion that they have a very large memory at their disposal even though the computer actually has a relatively small main memory. A virtual memory system provides a mechanism for translating program generated addresses into correct main memory locations. This is done dynamically while programs are being executed in the CPU. The translation or mapping is handled automatically by the hardware by means of mapping table.

3.5.2 Virtual Address and Physical Address

An address used by the programmer will be called virtual address and the set of such addresses are called the address space.

An address in main memory is called the physical address and the set of such locations are called the memory space.

The address space is allowed to be larger than the memory space in computers with virtual memory.

3.5.3.Memory Table for Mapping Virtual Address

Consider an example, a main memory capacity of 32K words. Fifteen bits are needed to specify a physical address in memory. Let the computer has auxiliary memory for storing 1024k words. Thus auxiliary memory has a capacity for storing information equivalent to the capacity of 32 main memories. Denoting the address space by N and the memory space by M ,N=1024k and M=32k

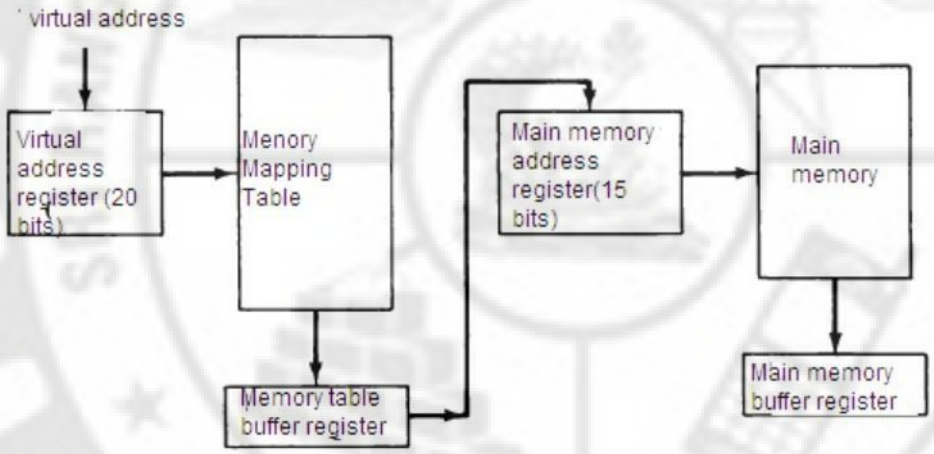


Fig 3.10 memory table for mapping a virtual address

The address field of an instruction code will consists of 20 bits but physical memory addresses must be specified with only 15 bits. A table is then needed as shown in Fig 3.10 to map a virtual address of 20 bits to a physical address of 15 bits. The mapping is a dynamic operation, which means that every address is translated immediately as a word is referenced by CPU. The mapping table may be stored in a separate memory or in main memory.

Address Mapping Using Pages

The information in the address space and the memory space are each divided into groups of fixed size. Let it be of 1k byte size. The physical memory group is known as blocks/page frames. The auxiliary memory group is known as page. i.e page size=block size

Let the main memory size is 32k bytes and divided into 32 blocks of 1kbyte size. The auxiliary memory size is 1024k bytes and divided into 1024 pages of 1k byte size. Portions of programs are moved from auxiliary memory to main memory in records equal to the size of a page.

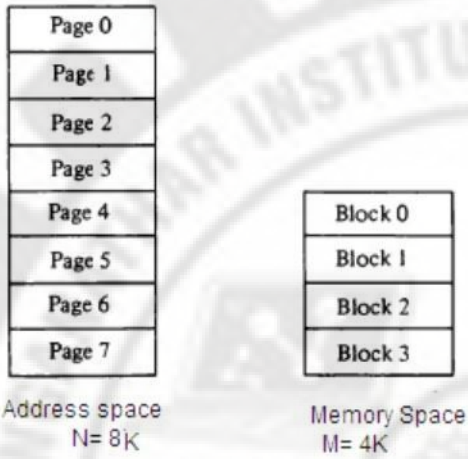


Fig 3.11 Address space and memory space split into groups of 1K words

Consider a computer with an address space of 8k and a memory space of 4k. if we split each into groups of 1k words we obtain eight pages and four blocks as shown in fig 3. 11.At any given time up to four pages of address space may reside in main memory in any one of the four blocks.

The mapping from address space to memory space is facilitated if each virtual address is considered to be represented by two numbers a page number address and a line within the page. In a computer with words per page k bits are used to specify a line address and the remaining high order bits of virtual space specify the page number.(i.e.) the line address in address space and memory space is the same, the only mapping required is from a page number to a block number.

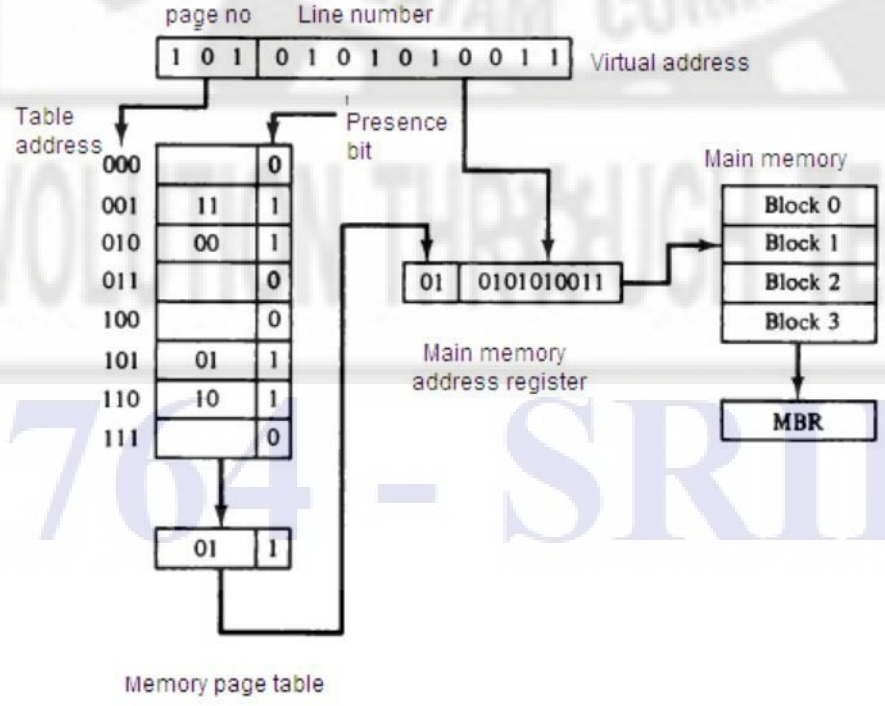


Fig 3.12 Memory table in a paged system

The organization of the memory mapping table in a paged system is shown in Fig 3.12.The address in the page table denotes the page number and the content of the word

gives the block number where that page is stored in main memory. A table shows that pages 1,2,5 and 6 are now available in main memory in blocks 3,0,1 and 2 respectively. A presence bit in each location indicates whether the page has been transferred from auxiliary memory into main memory. The CPU references a word in memory with virtual address of 13 bits. The three high order bits of the virtual address specify a page number and also a address for the memory page table. The content of word in the memory page table at the page number is read out into the memory table buffer register. if the presence bit is a 1, a block number thus read is transferred to the two high order bits of the main memory address register. The line number from the virtual address is transferred into the 10 low order bits of the memory address register. A read signal to main memory transfers the content of the word to the main memory buffer register ready to be used by the CPU. If the presence bit in the word read from the page table is 0 it signifies that the content of the word referenced by the virtual address does not reside in main memory.

Associative Memory Page Table

A more efficient way to organize the page table would be to construct it with a number of word equal to the number of blocks in main memory. In this way the size of the memory is reduced and each location is fully utilized. This method can be implemented by means of an associative memory with each word in memory containing page number together with its corresponding block number. The page field in each word is compared with the page number in the virtual address. If a match occurs the word is read from memory and its corresponding block number is extracted.

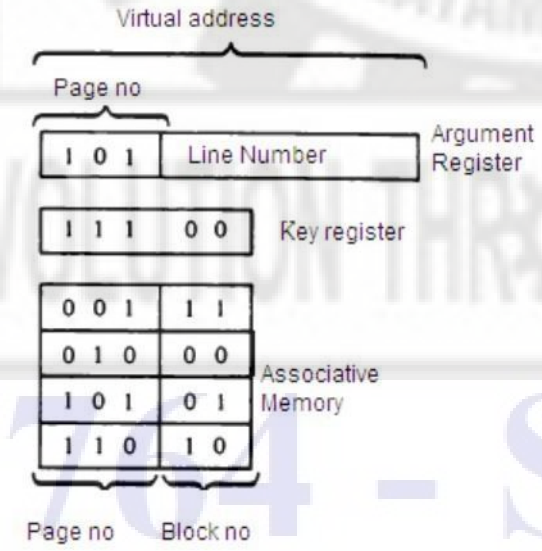


Fig 3.13 Associative memory page table

Consider a case of eight pages and four blocks as shown in fig 3.12. We replace the random access memory of four words as shown in fig 3.13 Each entry in the associative memory array consists of two fields. The first three bits specify a field for storing the page

number. The last two bits constitute a field for storing the block number. The virtual address is placed in the argument register. The page number bits in the argument register are compared with all page numbers in the page field of the associative memory. If the page number is found the 5 bit word is read out from memory. The corresponding block number being in the same word, is transferred to the main memory address register. If no match occurs, a call to the operating system is generated to bring the required page from auxiliary memory.

3.5.6.Page Replacement Technique

When a page fault occurs in a virtual memory system it signifies that the page referenced by the CPU is not in main memory. A new page is then transferred from auxiliary memory to main memory. If main memory is full it would be necessary to remove a page from memory block to make room for the new page. The policy for choosing pages to remove is determined from the replacement algorithm used. Two of the most common replacement algorithms used are the first in first out and the least recently used.

The FIFO algorithm selects for replacement the page that has been in memory the longest time. Each time a page is loaded into memory its identification number is pushed into a FIFO stack. FIFO will be full whenever memory has no more empty blocks. When a new page must be loaded the page least recently brought in is removed. The page to be removed is easily determined because its identification number is at the top of the FIFO stack. The FIFO replacement policy having advantage of being easy to implement. It has the disadvantage that under certain circumstances pages are removed and loaded from memory too frequently.

The LRU policy is more difficult to implement but has been more attractive. The algorithm can be implemented by associating a counter with every page that is in main memory. When a page is referenced its associated counter is set to zero. At fixed intervals of time the counters associated with all the pages presently in memory are incremented by 1. The least recently used page is the page with the highest count. The counters are often called aging registers as their count indicates their age, that is how long ago their associated pages have been referenced.

Summary

- Memory types—Register, SRAM, ROM, DRAM, Cache
- Memory Classification--- Primary, Secondary, Cache
- Semiconductor memories--- ROM,SRAM,DRAM
- Main memory------ROM and DRAM, moderate access time, medium cost, medium size, Secondary Memory --- Magnetic Disk, Magnetic tape, maximum access time, less cost, large size, Cache memory---- SRAM, minimum access time, high cost, small size,

- Cache memory under the principle of locality of reference and measured by hit ratio, Cache initialization----- valid bit inform about fresh data,Cache memory ---transfer the required data and instructions from main memory to cache by means of mapping, associative mapping, direct mapping and set associative mapping, Cache writing methods----write back and write through
- Memory management virtually showing secondary memory size as main memory size to the programmer.Virtual memory –address space, memory space, page size, frame size, Virtual memory---transfer the required data and instructions from secondary to primary by means of mappingVirtual memory mapping---Reducing secondary address into primary address by means of memory table Virtual memory Mapping types paging, associative
- Page replacement technique FIFO,LRU

Review Questions

2 marks

1. Why we need memory for a computer
2. What are the types of memory
3. Name some semiconductor memories
4. Define static RAM
5. Define DRAM
6. Define ROM
7. How many 512 kB RAM chips are required to construct a 1MB RAM capacity system
8. Why we need refreshing in DRAM
9. Which RAM is used as cache
10. What is the use of ROM chip in a system
11. What is bootstrap loader
12. Which is the basic unit for a SRAM
13. How many address lines are required for a 1024 KB RAM chip
14. What is the function of a decoder in a memory connection
15. What is sequential accessing give example for the sequential access type
16. What is random accessing give example for the random access type
17. Why secondary memory is needed
18. Mention any four secondary memory types
19. What is hard disk
20. Define a sector
21. Define a track
22. What is byte density in a disk
23. Where the density is high in a disk
24. Why we are moving to equal distribution of data
25. Why we have a cache memory
26. What is mapping technique
27. Define memory space
28. Define address space
29. Define virtual memory concept
30. Define paging
31. Define block
32. What is physical address
33. What is page replacement technique

3 marks

1. Discuss about various stages of memory
2. Why we need ROM for a system
3. Write the function of magnetic tape
4. How a magnetic disk works?
5. What are the different types of primary memory
6. Why a memory is called primary and secondary
7. What is the speed difference between primary and cache
8. How to initialize cache
9. Define write back
10. Define write through
11. What are the advantages and disadvantages in write back method
12. What are the advantages and disadvantages in write through method
13. What is the disadvantage of associative mapping in cache
14. Mention the disadvantage of direct mapping
15. Mention the disadvantage of set associative mapping
16. What is mapping table in virtual memory concept
17. Why we are going to virtual memory
18. Define hit ratio
19. Draw a memory map table for a physical memory of 4KB and virtual memory of 32KB
20. Mention the draw backs of memory map table
21. Why we are going towards associative page mapping
22. When we need page replacement in virtual memory
23. Discuss about FIFO page replacement technique
24. What is the use of page counter in LRU replacement technique

Five marks

1) Mention the different types of memory
2) Explain about RAM and ROM chip
3) Explain about memory address map
4) Draw a block diagram of connecting RAM and ROM together to form a main memory
5) Explain the operational principle of cache memory
6) Explain about direct mapping technique of cache memory
7) How a set associative mapping is operated and write about its advantages
8) Explain about virtual memory concept
9) How to map a virtual address using pages
10) Discuss about associative memory page table in virtual memory
11) Explain the page replacement techniques in detail.
12) How to convert auxiliary memory address into main memory address
13) Draw a memory table for main memory capacity of 8kB and secondary memory capacity of 16kB for a paged system
14) Draw direct mapping cache with block size of 16 words, cache memory size of 4kB and main memory size of 64Kb

UNIT IV. MICROPROCESSORS AND PARALLEL PROCESS

Objectives

- To study the block diagram of 8086 microprocessor and the various registers in it.
- To understand the concept of parallel processing.
- To understand the concept of pipelining.

INTRODUCTION

Microprocessor:

A computer whose entire CPU is contained on one integrated circuit. The important characteristics of a microprocessor are the widths of its internal and external address bus and data bus(and instruction),its clock rate and its instruction set. The CPU fetches instructions from memory, decodes and executes them. This may cause it to transfer data to or from memory or to activate peripherals to perform input or output. The address bus is the connection between the CPU and memory input/output which carry the address from/to which the CPU wishes to read or write. The number of bits of address bus determines the maximum size of memory which the processor can access. Register is one of a small number of high speed memory locations in a computer's CPU. RAM stores the data.

The CPU consists of :

Control Unit – which Generates control/timing signals and Controls decoding/execution of instructions.

ALU - Used during execution of instructions and mathematical and also logical operations.

Registers Instruction Pointer Counter, Instruction Register, Stack Pointer, General-Purpose Registers

Parallel computing, uses multiple processing elements simultaneously to solve a problem. The processing elements can be diverse and include resources such as a single computer with multiple processors, several networked computers, specialized hardware, or any combination of the above. The parallel systems were classified into SISD, SIMD, MISD and MIMD.

Pipelining is a technique of decomposing a sequential process into sub operations with each sub process being executed in a special dedicated segment that operates concurrently with all other segments. Arithmetic Pipeline and Instruction pipeline are the two different types of pipelines in computer architecture.

BLOCK DIAGRAM OF INTEL 8086

The 8086 CPU is divided into two independent functional units:

1. Bus Interface Unit (BIU)

2. Execution Unit (EU)

Figure 4.1 shows the block diagram of 8086

Features of 8086 Microprocessor:

1. Intel 8086 was launched in 1978.

2. It was the first 16-bit microprocessor.

3. This microprocessor had major improvement over the execution speed of 8085.

4. It is available as 40-pin Dual-Inline-Package (DIP).

5. It is available in three versions:

a. 8086 (5 MHz)

b. 8086-2 (8 MHz)

c. 8086-1 (10 MHz)

6. It consists of 29,000 transistors.

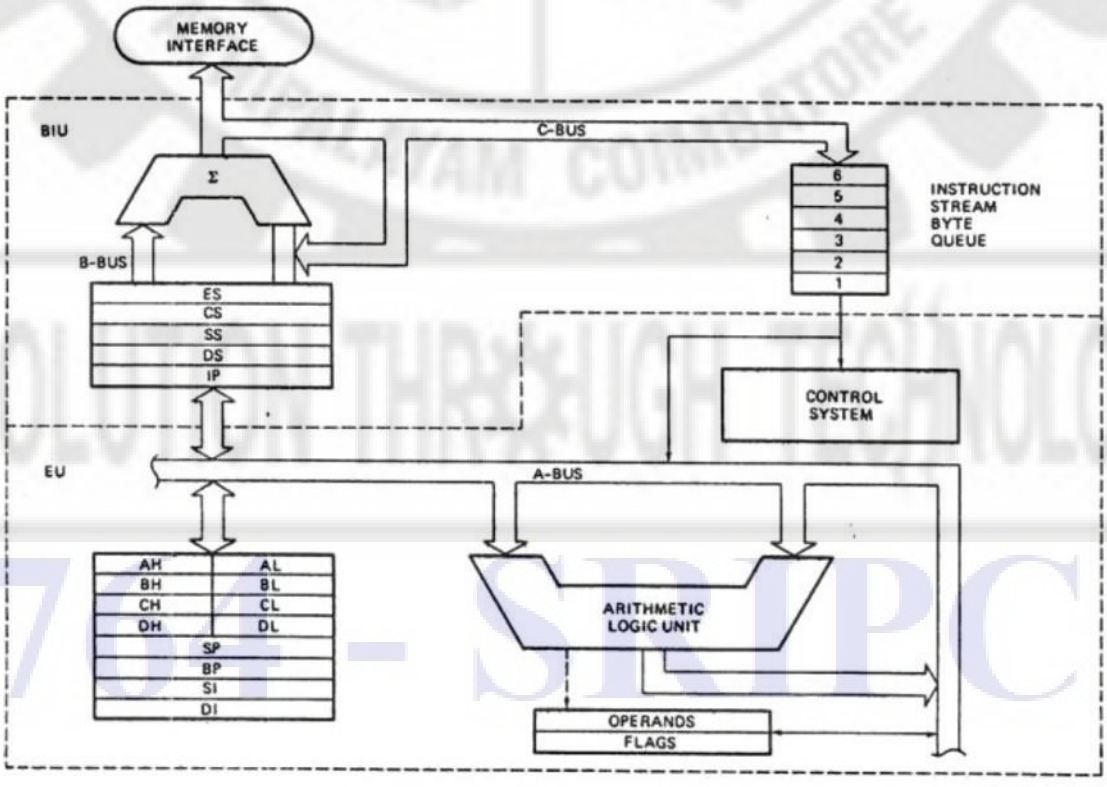


Figure 4.1 **Block Diagram of Intel 8086**

Bus Interface Unit (BIU)

The function of BIU is to:

1. Fetch the instruction or data from memory.

2.Write the data to memory.

3.Write the data to the port.

4.Read data from the port.

Instruction Queue

1. To increase the execution speed, BIU fetches as many as six instruction bytes ahead to time
from memory.

2. All six bytes are then held in first in first out 6 byte register called instruction queue.

3. Then all bytes have to be given to EU one by one.

4. This pre fetching operation of BIU may be in parallel with execution operation of EU, which
improves the speed execution of the instruction.

Execution Unit (EU)

1. The functions of execution unit are:
2. To tell BIU where to fetch the instructions or data from.
3. To decode the instructions.
4. To execute the instructions.

The EU contains the control circuitry to perform various internal operations. A decoder in EU decodes the instruction fetched from memory to generate different internal or external control signals required to perform the operation. EU has 16-bit ALU, which can perform arithmetic and logical operations on 8-bit as well as 16-bit.

General Purpose Registers of 8086

These registers can be used as 8-bit registers individually or can be used as 16-bit in pair to have AX,BX, CX, and DX.

1. AX Register: AX register is also known as accumulator register that stores operands for arithmetic operation like divided, rotate.

2. BX Register: This register is mainly used as a base register. It holds the starting base location of a memory region within a data segment.

3. CX Register: It is defined as a counter. It is primarily used in loop instruction to store loop counter.

4. DX Register: DX register is used to contain I/O port address for I/O instruction.

Segment Registers

Additional registers called segment registers generate memory address when combined with other in the microprocessor. In 8086 microprocessor, memory is divided into 4 segments as follows:

1. Code Segment (CS): The CS register is used for addressing a memory location in the Code Segment of the memory, where the executable program is stored.

2. Data Segment (DS): The DS contains most data used by program. Data are accessed in the Data Segment by an offset address or the content of other register that holds the offset address.

3. Stack Segment (SS): SS defined the area of memory used for the stack.

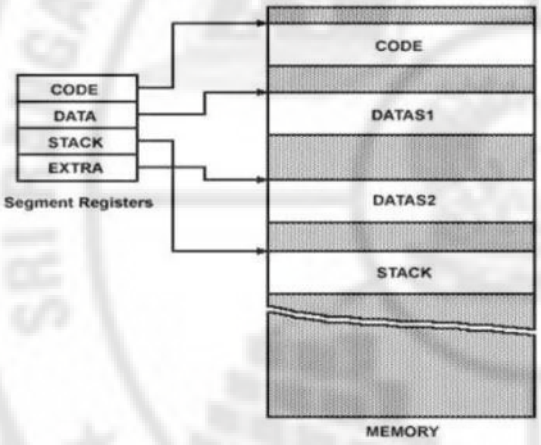


Figure 4.2 Memory Segments of 8086

Address:

Group of bits which are used to identify the locations in the memory is called address.

Effective address:

All memory locations within a segment are relative to the segment's starting address. The distance in bytes from the segment address to another location within the segment is expressed as an offset of displacement.

Effective address = segment starting address+ displacement

Segment address:

A segment is a special area in a program that begins on a paragraph boundary. Segment address represents the starting address or base address of each segment.

Flag Registers of 8086

Flag register in EU is of 16-bit and is shown in fig. 4.3:

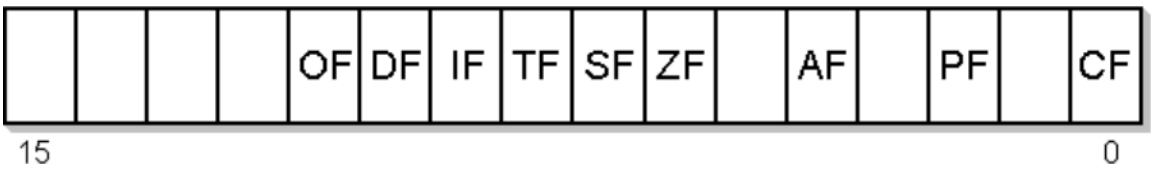


Figure 4.3 Flag Register of 8086

Flag Register determines the current state of the processor. They are modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program. 8086 has 9 flags and they are divided into two categories:

1. Conditional Flags

2. Control Flags

Conditional Flags

Conditional flags represent result of last arithmetic or logical instruction executed.

Conditional flags are as follows:

· **Carry Flag (CF):** This flag indicates an overflow condition for unsigned integer arithmetic. It is also used in multiple-precision arithmetic.

· **Auxiliary Flag (AF):** If an operation performed in ALU generates a carry/borrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), the AF flag is set i.e. carry given by D3 bit to D4 is AF flag. This is not a general-purpose flag, it is used internally by the processor to perform Binary to BCD conversion.

· **Parity Flag (PF):** This flag is used to indicate the parity of result. If lower order 8-bits of the result contains even number of 1's, the Parity Flag is set and for odd number of 1's, the Parity

Flag is reset.

· **Zero Flag (ZF):** It is set; if the result of arithmetic or logical operation is zero else it is reset.

· **Sign Flag (SF):** In sign magnitude format the sign of number is indicated by MSB bit. If the result of operation is negative, sign flag is set.

Overflow Flag (OF): It occurs when signed numbers are added or subtracted. An OF indicates that the result has exceeded the capacity of machine.

Control Flags

Control flags are set or reset deliberately to control the operations of the execution unit. Control flags are as follows:

1. Trap Flag (TP):

a. It is used for single step control.

b. It allows user to execute one instruction of a program at a time for debugging.

c. When trap flag is set, program can be run in single step mode.

2. Interrupt Flag (IF):

a. It is an interrupt enable/disable flag.

b. If it is set, the maskable interrupt of 8086 is enabled and if it is reset, the interrupt is disabled.

c. It can be set by executing instruction sit and can be cleared by executing CLI instruction.

3. Direction Flag (DF):

a. It is used in string operation.

b. If it is set, string bytes are accessed from higher memory address to lower memory address.

c. When it is reset, the string bytes are accessed from lower memory address to higher memory address.

4.1.1 Applications of Microprocessor

Microprocessors are applicable to a wide range of information processing tasks, ranging from general computing to real -time monitoring systems. The microprocessor facilitates new ways of communication and how to make use of the vast information available online and offline both at home and in workplace. Most electronic devices including everything from computers, remote controls, washing machines, microwaves and cell phones to iPods and more -- contain a built- in microprocessor. Microprocessors are at the core of personal computers, laptops, mobile phones and complex military and space systems.

1. General purpose applications

i) Single board micro computers

Single board microcomputers are simple and cheaper. They have the minimum possible software and hardware configuration. They are used to train the students. They are also used to build small computer based systems.

ii) Personal Computers

The home computer made up of a 8 bit microprocessor are used for playing video games and learning simple programs. The 16 bit computers are used for word processing, payroll, business accounts, etc.

iii) Super Minis and CAD

32 bit processors are used to build powerful microcomputers. Their performance is better than mainframe and mini-computers. These systems are used in Engineering side as computer aided design machines.

2. Special purpose application.

i) Instrumentation

The microprocessor based instrument can be made intelligent using feature programmability. Microprocessors can be used as controllers in various instruments. They are also used in medical instruments to measure blood pressure and temperature.

ii) Control

Microprocessor controllers are now available in home appliances such as microwave oven, washing machine. In industry, microprocessor are used in controlling various process.

iii) Communication

In the telephone industry, microprocessors are used in digital telephone sets, telephone exchanges and modems. They are also used in railway reservation system at the national level and air reservation system at international level. Satellite communication systems, mobile phones and televisions are also using microprocessors.

iv) Office Automation and Publication

With the availability of inexpensive and user friendly microcomputers along with wide range of software packages, office works are computerized. Microcomputers are used in office to perform word processing, spreadsheet operations, storage and retrieval of huge information. In publishing houses microprocessor based systems are used for making automatic photo copies. Microprocessor based LASER printers are used to achieve good speed in printing.

PARALLEL PROCESSING

Types of Parallel Processor Systems

Parallel computing, uses multiple processing elements simultaneously to solve a problem. This is accomplished by breaking the problem into independent parts so that each processing element can execute its part of the algorithm simultaneously with the others.

Michael J. Flynn created one of the earliest classification systems for parallel (and sequential) computers and programs, now known as Flynn's taxonomy. Flynn classified programs and computers by whether they were operating using a single set or multiple sets of instructions, and whether or not those instructions were using a single set or multiple sets of data.

	Single instruction	Multiple instruction
Single data	SISD	MISD
Multiple data	SIMD	MIMD

Figure 4.4 Flynns Taxonomy

Flynn proposed the following categories of computer systems:

• **Single instruction, single data (SISD) stream:** A single processor executes a single instruction stream to operate on data stored in a single memory. Uniprocessors fall into this category.

• **Single instruction, multiple data (SIMD) stream:** A single machine instruction controls the simultaneous execution of a number of processing elements on a lockstep basis. Each processing element has an associated data memory, so that each instruction is executed on a different set of data by the different processors. Vector and array processors fall into this category.

• **Multiple instruction, single data (MISD) stream:** A sequence of data is transmitted to a set of processors, each of which executes a different instruction sequence.This structure is not commercially implemented.

• **Multiple instruction, multiple data (MIMD) stream:** A set of processors simultaneously execute different instruction sequences on different data sets. SMPs, clusters, and NUMA systems fit into this category.

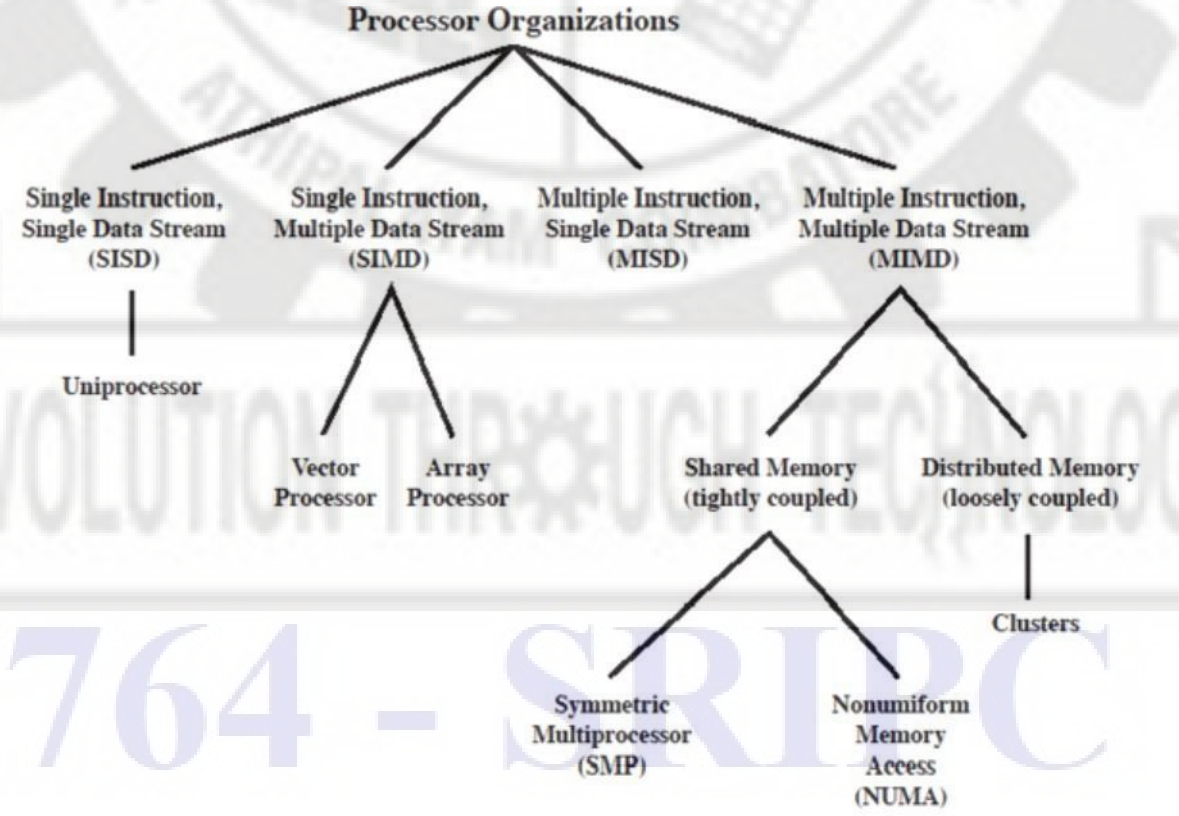


Figure 4.5 A Taxonomy of Parallel Processor Architectures

Parallel OrganizationsSISD

Figure 4.6a shows the structure of an SISD. There is some sort of control unit (CU) that provides an instruction stream (IS) to a processing unit (PU). The processing unit operates on a single data stream (DS) from a memory unit (MU).

Figure 4.6 (a) SISD

SIMD

In an SIMD, there is still a single control unit, now feeding a single instruction stream to multiple PUs. Each PU may have its own dedicated memory (illustrated in Figure 4.6 b), or there may be a shared memory.

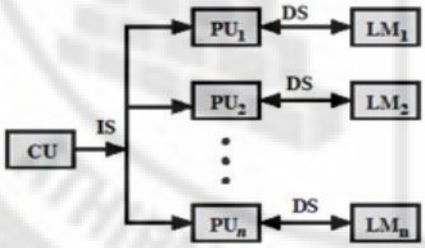


Figure 4.6 (b) SIMD (with distributed memory)

MIMD

In the MIMD, there are multiple control units, each feeding a separate instruction stream to its own PU. The MIMD may be a shared-memory multiprocessor (Figure 4.3.3c) or a distributed-memory multicomputer (Figure 4.3.3d).

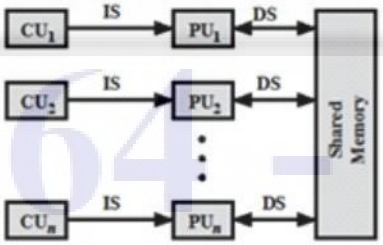


Figure 4.6 (c) MIMD (with shared memory)

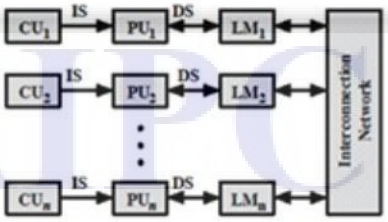


Figure 4.6 (d) MIMD (with distributed memory)

PIPELINING

Pipelining is a technique of decomposing a sequential process into sub operations with each sub process being executed in a special dedicated segment that operates concurrently with all other segments. A pipeline is like a collection of processing segments through which binary information flows. Each segment performs partial processing .The result obtained from the computation in each segment is transferred to the next segment in the pipeline. The final result is obtained after the data has passed through all the segments. It is the characteristics of the pipeline that several computations can be in progress in distinct segments at the same time.

There are two areas of computer design where pipeline organization is applicable:

- An arithmetic pipeline divides an arithmetic operation into sub operations for execution in the pipeline segments.
- An instruction pipeline operates on a stream of instructions by overlapping the fetch, decode and execute phases of the instruction cycle.

General operation

Any operation that can be decomposed into a sequence of sub operations of about the same complexity can be implementd by a pipeline processor. The general structure of a pipeline is illustrated in the figure 4.7. The operands pass through all four segments in a fixed sequence. The segments are separated by registers R_i that hold the intermediate result s between the stages. Information flows through a common clock applied to all registers simultaneously.

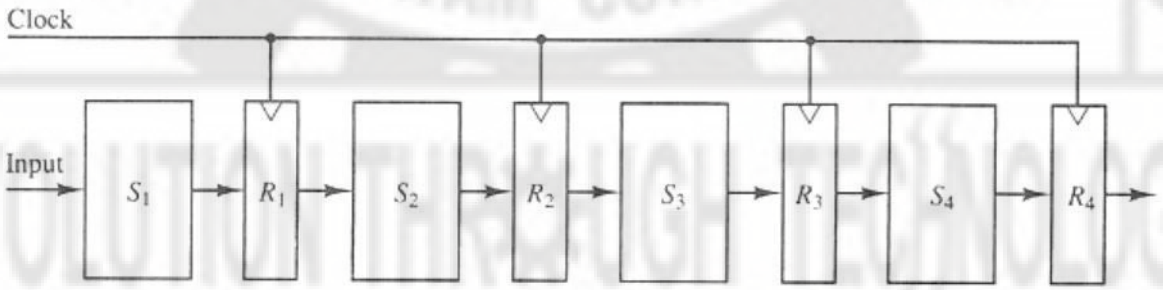


Figure 4.7 A Four segment pipeline

ARITHMETIC PIPELINE

Pipeline arithmetic units are usually found in high speed computers. They are used to implement floating –point operations, multiplication of fixed point numbers and similar computations encounterd in scientific problem. Let us consider an example for floating point addition and subtraction. A and B are two fractions that represent the mantissa and a and b are the exponents.

The floating point addition and subtraction can be performed in four segments as shown in figure 4.8. The sub operations that are performed in the four segments are:

1. Compare the exponents.
2. Align the mantissa'
3. Add or subtract the mantissas.
4. Normalize the results.

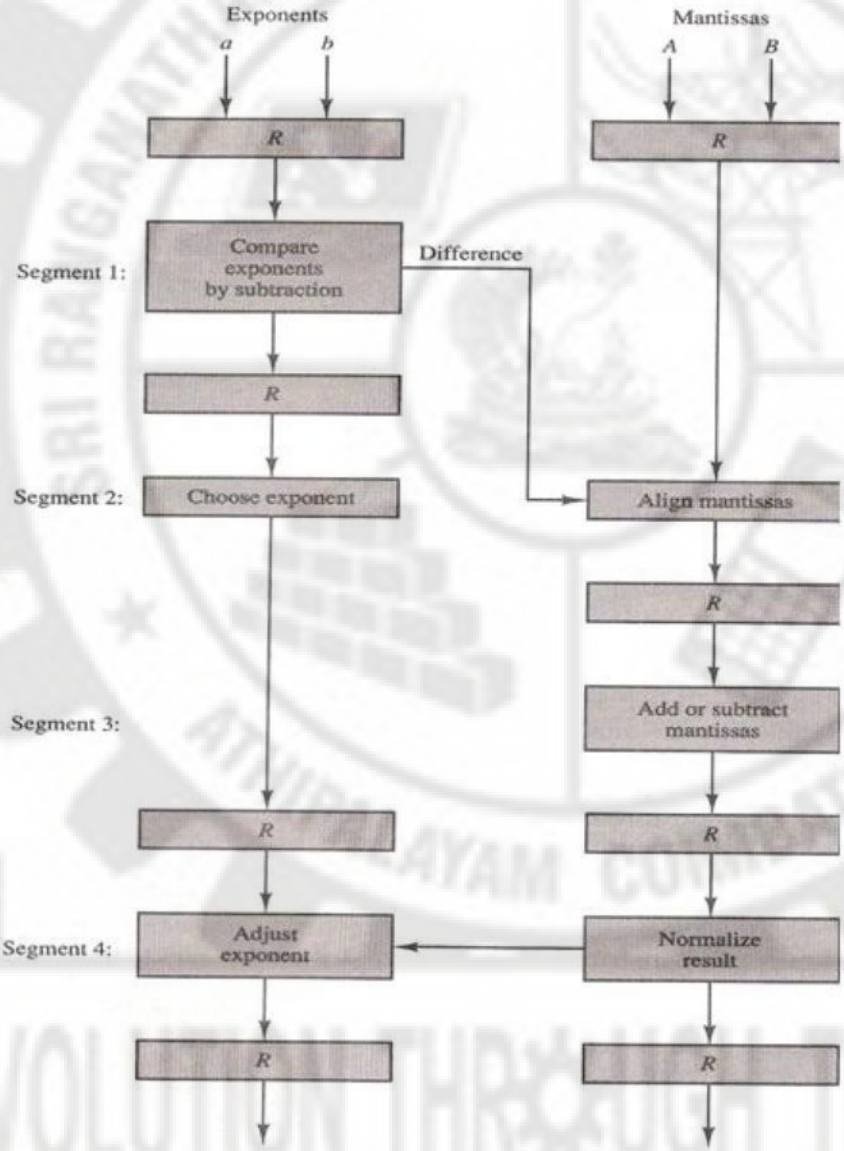


Figure 4.8 Pipeline for Floating Point Addition and Subtraction.

The following numerical example illustrates the arithmetic pipeline. Consider two normalized floating point numbers

$$X = 0.9504 \times 10^3$$

$$Y = 0.8200 \times 10^2$$

The two exponents are subtracted in the first segment to obtain 3-2 = 1. The larger exponent 3 is chosen as the exponent of the result. The next segment shifts the mantissa of Y to the right to obtain

$$X = 0.9504 \times 10^3$$

$$Y = 0.0820 \times 10^3$$

This aligns the two mantissas under the same exponent. The addition of two mantissas in segment 3 produces the sum

$$Z = 1.0324 \times 10^3$$

The sum is adjusted by normalizing the result so that it has a fraction with a non zero first digit. This is done by shifting the mantissa once to the right and incrementing the exponent by one to obtain the normalized sum.

$$Z = 0.10324 \times 10^4$$

The comparator, shifter, adder-subtractor, incrementer and decrementer in the floating point pipeline are implemented with combinatorial circuits.

Instruction Pipeline

An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments. This causes the instruction fetch and execute phases to overlap and perform simultaneous operations. In case of a branch instruction the pipeline is emptied and all the instructions that have been read from memory after the branch instruction must be discarded.

Computers with complex instructions require other phases in addition to the fetch and execute to process an instruction completely. Computer processes each instruction with the following sequence of steps:

1.Fetch the instruction from memory.

2.Decode the instruction.

3.Calculate the effective address.

4.Fetch the operands from memory.

5.Execute the instruction.

6.Store the result in the proper place.

Four Segment Instruction pipeline

Figure 4.9 shows how the instruction cycle in the CPU can be processed with a four-segment pipeline. While an instruction is being executed in segment 4, the next instruction in sequence is busy fetching an operand from memory in segment 3.The effective address may be calculated in a separate arithmetic circuit for the third instruction and

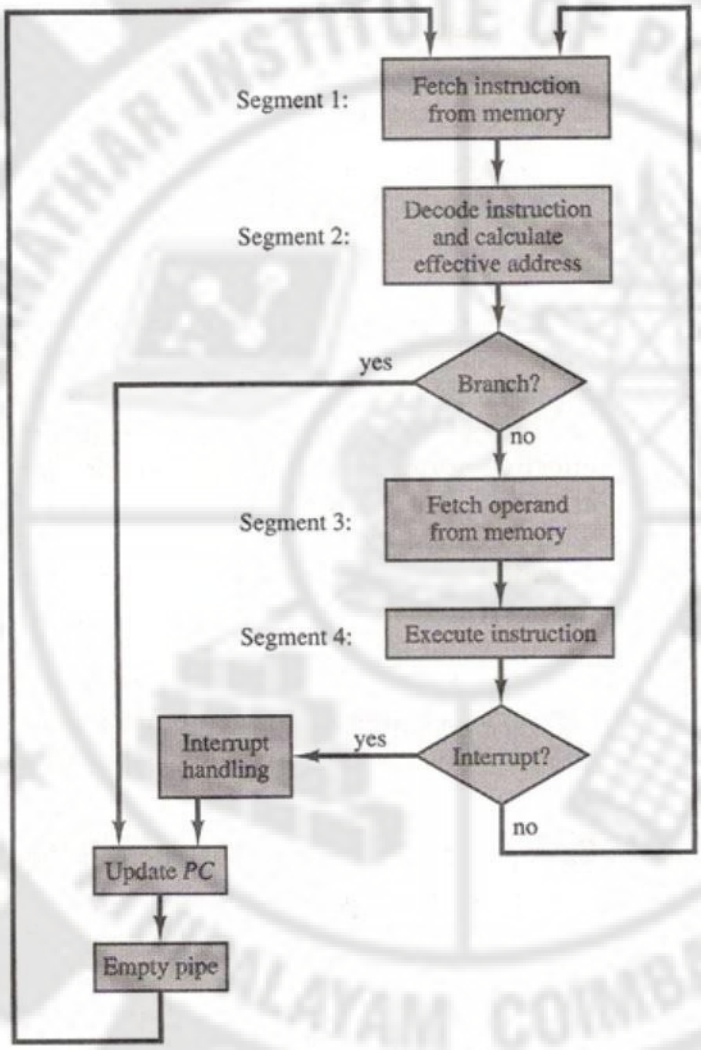


Figure 4.9 A four segment CPU pipeline

.

whenever memory is available the fourth and subsequent instructions can be fetched and placed in an instruction FIFO.Thus upto four suboperations in the instruction cycle can overlap and upto four different instructions can be in progress of processed at the same time Once in a while an instruction in the sequence may be a program control type that causes a branch out of normal sequence. In that case the pending operations in the last two segments are completed and all information stored in the instruction buffer is deleted. The pipeline then restarts from the new address stored in the program counter. Similarly, an interrupt request will cause the pipeline to empty and start again from a new address value.

Super Scalar Pipeline

Superscalar pipelines are :

- Parallel pipelines which can initiate the processing of multiple instructions in every machine cycle.

- Diversified pipelines in employing multiple and heterogeneous functional units in execution stages.
- Dynamic pipelines which can achieve the best performance without re-ordering of instructions by the compiler.

Summary

8086 has two blocks BIU and EU.The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands. The instruction bytes are transferred to the instruction queue. EU executes instructions from the instruction system byte. Both units operate asynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as Pipelining. This results in efficient use of the system bus and system performance. BIU contains Instruction queue, Segment registers, Instruction pointer, Address adder.EU contains Control circuitry, Instruction decoder, ALU,Pointer and Index register, Flag register.In the chapter The Flynns taxonomy of parallel computers is discussed. The various parallel computer organizations like SISD,SIMD,MISD,MIMD is discussed. The concept of pipelining and the different types of pipelining with block diagram is discussed . A pipeline is like a collection of processing segments through which binary information flows. The main areas of using pipelines in computer architecture is Arithmetic pipeline and Instruction pipeline.

Review Questions

Two marks

1. List any 2 features of 8086.
2. What are the main units of 8086 microprocessor.
3. What is the function of Bus interface unit.
4. What is the function of Execution unit.
5. List down the various general purpose registers of 8086.
6. List down the various segment registers.
7. What is a flag register.
8. What is parallel computing.
9. List down the Flynn's classifications of computer.
10. What is pipelining.
11. What are the two different types of pipelining.
12. What is an arithmetic pipeline.
13. What is an instruction pipeline.

3 marks

1. Explain the importance of Instruction queue in 8086.
2. Explain the various general purpose registers of 8086.
3. Explain about the various segment registers of 8086.
4. Draw the flag register. Explain about the various settings of the flags.
5. What are the special purpose applications of a micro-processor.
6. Draw the taxonomy of parallel computers.
7. Explain a four stage pipeline with block diagram.
8. List down the various steps in the processing of an instruction.
9. List down the steps in an adder/subtractor pipeline.
10. What are superscalar pipelines.

Five/Ten marks.

1. Explain the architecture of 8086 with a neat block diagram.
2. Explain the various types of flags in 8086 in detail.
3. Explain the various types of applications in a microprocessor.
4. Explain the various types of parallel computer organizations.
5. Explain the various stages of an arithmetic pipeline in detail with block diagram.
6. Explain the various stages of an instruction pipeline with block diagram.

UNIT V. ARCHITECTURE AND CONCEPTS OF ADVANCED PROCESSORS

Objectives

- To understand what a Symmetric multiprocessor is, to study the general organization of a Symmetric Multiprocessor and to study how Z990 is organized.
- To understand Multithreading, to study the difference between a process and a thread, to understand the concept of clusters and to study the various cluster classifications and configurations.
- To understand what NUMA is, to study the difference between NUMA, SMP and Clusters to study the CC-NUMA organisation, to understand what vector computation is and to study the various approaches to vector computation.
- To explain a multicore processor, to understand the multicore organizations, and to study the organization of a Core2 Duo and Core i7 processor.

INTRODUCTION

A traditional way to increase system performance is to use multiple processors that can execute in parallel to support a given workload. The most common multiple-processor organizations are Symmetric Multi Processors, Clusters and NonUniform Memory Access (NUMA).

Clustering is an alternative to symmetric multiprocessing as an approach to providing high performance and high availability and is particularly attractive for server applications. A cluster is a group of interconnected, whole computers working together as a unified computing resource. Various types of cluster configurations and classifications are present. An alternative approach, which allows for a high degree of performance without increasing circuit complexity or power consumption, is called multithreading. In computer architecture, multithreading is the ability of a central processing unit (CPU) or a single core in a multi-core processor to execute multiple processes or threads concurrently, as supported by the operating system.

The two common approaches to providing a multiple-processor system to support applications are SMPs and clusters. Another approach, known as NonUniform Memory Access (NUMA) is where all processors have access to all parts of memory.

Maths problems involving physical processes present different difficulties for computation. For example, aerodynamics, seismology, meteorology, Continuous field simulation, High precision Repeated floating point calculations on large arrays of numbers etc. Supercomputers handle these types of problems. Array processors are alternatives to

supercomputer. Configured as peripherals to mainframe & minicomputers they run vector portion of problems.

A multicore computer, or chip multiprocessor, combines two or more processors on a single computer chip. The multicore architecture poses challenges to software developers to exploit the capability for multithreading across multiple cores.

Symmetric Multiprocessors

SMP (**symmetric multiprocessing**) is the processing of programs by multiple processors that share a common operating system and memory.

The characteristics of a Symmetric Multiprocessor are:

- There are two or more similar processors of equal capability.
- These processors share the same main memory and I/O facilities.
- The memory access time of the processors is almost the same for each processor.
- All processors can perform the same function .
- The system is controlled by a single operating system .

Advantages of a SMP organization over a Uniprocessor organization

	Uniprocessor	Multiprocessor
Performance	Less Performance	Greater Performance
Availability	Failure of a single processor halts the system	Failure of a single processor will not halt the system.
Incremental Growth	Performance cannot be enhanced.	Performance can be enhanced by adding a processor.
scaling	Varied performance and price is not possible	Varied performance and price is possible

The operating system must provide tools and functions to exploit the parallelism in an SMP system. An attractive feature of an SMP is that the existence of multiple processors is transparent to the user. The operating system takes care of scheduling of threads or processes on individual processors and of synchronization among processors.

Organisation

Figure 5.1 depicts the organization of a multiprocessor system. There are two or more processors. Each processor is self-Contained, including a control unit, ALU, registers, and one or more levels of cache.

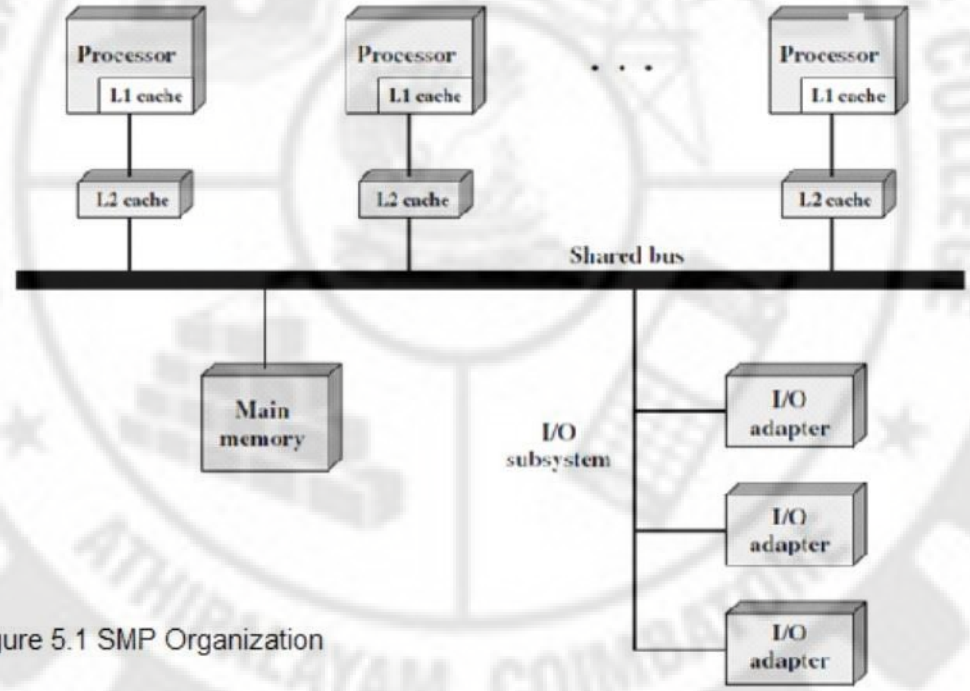


Figure 5.1 SMP Organization

Each processor has access to a shared main memory and the I/O devices through some form of interconnection mechanism. The processors can communicate with each other through memory. In some configurations, each processor has its own private main memory and I/O channels in addition to the shared resources. The most common organization for personal computers, workstations, and servers is the time-shared bus. The time-shared bus is the simplest mechanism for constructing a multiprocessor system (Figure 5.1).The structure and interfaces are basically the same as for a single-processor system that uses a bus interconnection.

Time sharing is achieved as follows:

When one module is controlling the bus, other modules are locked out and suspend operation until bus access is achieved. In an SMP , there are multiple processors as well as multiple I/O processors all attempting to gain access to one or more memory modules via the bus.

Features of the bus organization

• **Simplicity:** This is the simplest approach to multiprocessor organization. The physical interface and the addressing, arbitration, and time-sharing logic of each processor remain the same as in a single-processor system.

• **Flexibility:** It is easy to expand the system by attaching more processors to the bus.

• **Reliability:** The failure of any attached device will not cause failure of the whole system.

The main drawback to the bus organization is performance. All memory references pass through the common bus. Thus, the bus cycle time limits the speed of the system. To improve performance, workstation and PC SMPs are equipped with two levels of cache, with the L1 cache internal (same chip as the processor) and the L2 cache either internal or external. Some processors have L3 cache as well.

A MAINRAME SMP

A new approach to bus interconnection is used for a recent implementation of the IBM Z Series mainframe family called the Z990. This family of systems spans a range from a uniprocessor with one main memory card to a high-end system with 48 processors and 8 memory cards.

The key components of the configuration as shown in Figure 5.2 are:

• **Dual-core processor chip:** Each processor chip includes two identical central processors (CPs).The CP is a CISC superscalar microprocessor, in which most of the instructions are hardwired and the rest are executed by vertical microcode.Each CP includes a 256-KB L1 instruction cache and a 256-KB L1 data cache.

• **L2 cache:** Each L2 cache contains 32 MB. The L2 caches are arranged in clusters of five.

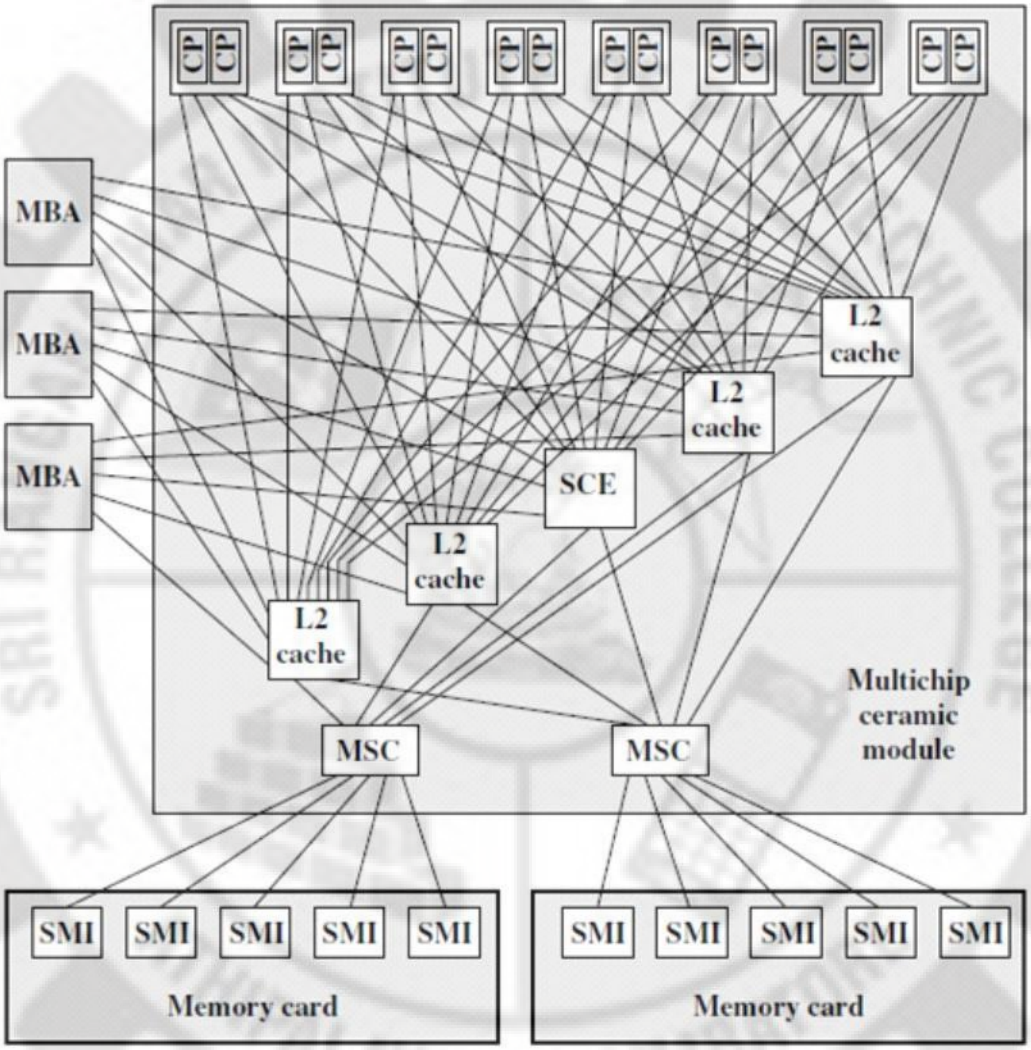
• **System control element (SCE):** The SCE helps in system Communication, and has a central role in maintaining cache coherence.

• **Main store control (MSC):** The MSCs interconnect the L2 caches and the main memory.

• **Memory card:** Each card holds 32 GB of memory. The maximum configurable memory consists of 8 memory cards for a total of 256 GB. Memory cards interconnect to the MSC via synchronous memory interfaces (SMIs).

• **Memory bus adapter (MBA):** The MBA provides an interface to various types of I/O channels. Traffic to/from the channels goes directly to the L2 cache.

The microprocessor in the z990 executes instructions in strict architectural order. However, it makes up for this by having a shorter pipeline and much larger caches and TLBs compared with other processors, along with other performance-enhancing features.



CP = central processor
MBA = memory bus adapter
MSC = main store control
SCE = system control element
SMI = synchronous memory interface

Figure 5.2 IBM z990 Multiprocessor Structure

The z990 system comprises of one to four **books**. Each book is a pluggable unit containing up to 12 processors with up to 64 GB of memory, I/O adapters, and a system control element (SCE) that connects these other elements. The SCE within each book contains a 32-MB L2 cache, which serves as the central coherency point for that particular book. Both the L2 cache and the main memory are accessible by a processor or I/O adapter within that book or any of the other three books in the system. The SCE and L2 cache chips also connect with corresponding elements on the other books in a ring configuration.

MULTI THREADING AND CLUSTERS

Designers have pursued the goal of increased performance on two fronts: increasing clock frequency and increasing the number of instructions executed or, more properly, the number of instructions that complete during a processor cycle. An alternative approach, which allows for a high degree of instruction-level parallelism without increasing circuit complexity or power consumption, is called multithreading. In essence, the instruction stream is divided into several smaller streams, known as threads, such that the threads can be executed in parallel. In computer architecture, multithreading is the ability of a central processing unit (CPU) or a single core in a multi-core processor to execute multiple processes or threads concurrently, as supported by the operating system.

5.2.1 Implicit and Explicit Multithreading

5.2.1.1Threads and Processes

Process: An instance of program running on computer.

Characteristics of a process:

A process has two key characteristics

- **Resource ownership:** A process is allocated control or ownership of resources, such as main memory, I/O channels, I/O devices, and files.
- **Scheduling/execution:** A process has an execution state (Running, Ready, etc.) and a dispatching priority and is scheduled and dispatched by the operating system.

Process switch is an operation that switches the processor from one process to another. This operation saves all the process control data, registers, and other information of the first process and replaces them with the process information for the second.

Thread: Is a dispatchable unit of work within the process with the following characteristics:

- Includes processor context (which includes the program counter and stack pointer) and data area for stack.
- Thread executes sequentially.
- Interruptible: Processor can turn to another thread.

Thread switch is switching processor between threads within same process. Most modern operating systems, such as Linux, other versions of UNIX, and Windows, support thread.

Process vs Threads

	Process	Threads
1.	A process is concerned with both scheduling/ execution and resource ownership	A thread is concerned with scheduling and execution

2.	Each process has its own resource. Hence process switch is more time consuming.	The multiple threads within a process share the same resources. Hence thread switch is less time consuming.

Types of Thread:

1. Explicit Thread
2. Implicit Thread

Explicit thread

Explicit Multithreading concurrently executes instructions from different explicit threads. This Interleaves instructions from different threads on shared pipelines or parallel execution on parallel pipelines.

Explicit threads are further classified as

- User Level Threads - visible to the application program
- Kernel Level Threads - visible only to the operating system

Implicit Thread

Implicit multithreading concurrently executes multiple threads extracted from single sequential program. Implicit threads are defined statically by compiler or dynamically by hardware. Most modern operating systems, such as Linux, new versions of UNIX, and Windows, support thread.

Approaches To Explicit MultiThreading

At minimum, a multithreaded processor must provide a separate program counter for each thread of execution to be executed concurrently. The processor treats each thread separately and may use a number of techniques for optimizing single-thread execution, including branch prediction, register renaming, and superscalar techniques. The result of this is thread-level parallelism. Broadly speaking, there are four principal approaches to multithreading:

Interleaved multithreading:

This is also known as fine-grained multi threading. The processor deals with two or more thread contexts at a time, switching from one thread to another at each clock cycle. If a thread is blocked because of data dependencies or memory latencies. that thread is skipped and a ready thread is executed.

Blocked multithreading:

This is also known as coarse-grained multithreading. The instructions of a thread are executed successively until an event occurs that may cause delay, such as a cache miss.

This event induces a switch to another thread. This approach is effective on an in-order processor that would stall the pipeline for a delay event such as a cache miss.

Simultaneous multithreading (SMT):

Instructions are simultaneously issued from multiple threads to the execution units of a superscalar processor. This combines the wide superscalar instruction issue capability with the use of multiple thread contexts.

Chip multiprocessing:

In this case, the entire processor is replicated on a single chip and each processor handles separate threads. The advantage of this approach is that the available logic area on a chip is used effectively without depending on ever-increasing complexity in pipeline design. This is referred to as multicore.

Clusters

Clusters are collection of independent whole uniprocessors or SMPs usually called nodes interconnected to form a cluster. They

- Work together as unified resource -Illusion of being one machine
- Communicate via fixed path or network connections

Benefits that can be achieved with clustering:

• **Absolute scalability:** It is possible to create large clusters that far surpass the power of even the largest standalone machines. A cluster can have tens, hundreds, or even thousands of machines, each of which is a multiprocessor.

• **Incremental scalability:** A cluster is configured in such a way that it is possible to add new systems to the cluster in small increments. Thus, a user can start out with a modest system and expand it slowly.

• **High availability:** Because each node in a cluster is a standalone computer, the failure of one node does not mean loss of service. Mostly fault tolerance is handled automatically in software.

• **Superior price/performance:** By using commodity building blocks, it is possible to put together a cluster with equal or greater computing power than a single large machine, at much lower cost.

Cluster Configurations:

There are two types of cluster configurations

(i) High speed link configuration with no shared disk. This (Figure 5.3a) uses high speed links between nodes. The link can be a LAN that is shared with other computers that are not part of the cluster or the link can be a dedicated interconnection facility.

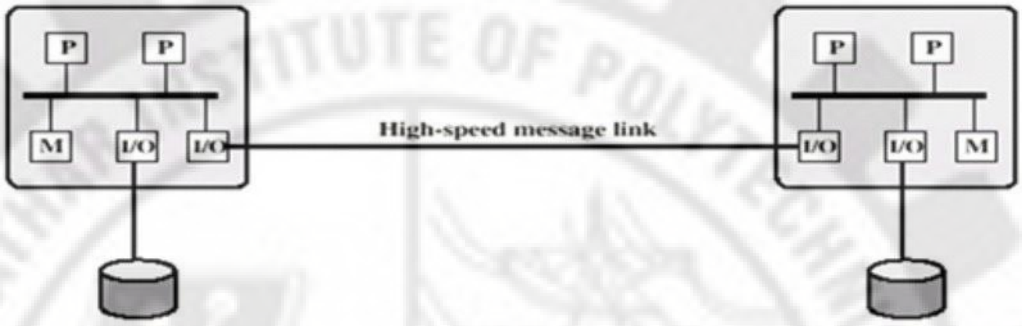


Figure 5.3 a

ii. The second one is a shared-disk cluster configuration(Figure 5.3 b). Here, there is a message link between nodes. There is a disk subsystem (RAID subsystem) that is directly linked to multiple computers within the cluster.

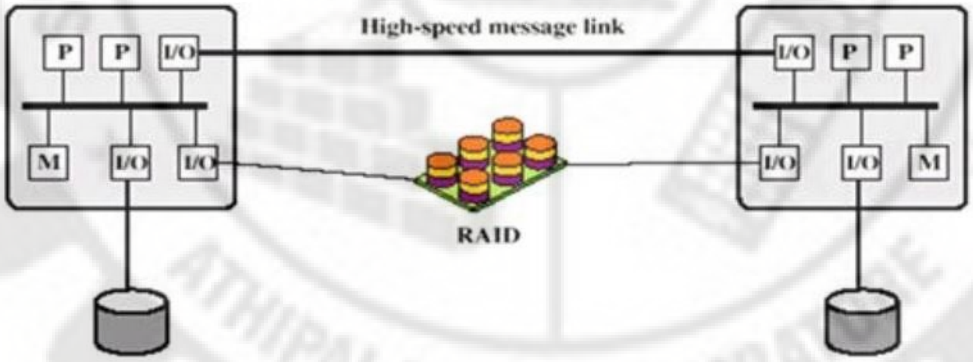


Figure 5.3b

Cluster Classification

Three classifications of clustering can be identified: separate servers, shared nothing, and shared memory.

Separate server:

This has its own disks and there are no disks shared between systems . The main features are:

- Data is continuously copied from primary to secondary server.
- High availability.
- High network and server overhead due to copying operations.

Shared Nothing Servers

Here the Servers are cabled to the same disks, but each server owns its disks. If one server fails, its disks are taken over by the other server. The main features are:

- Reduced network and server overhead due to elimination of copying operations.
- Usually requires disk mirroring or RAID technology to compensate for risk of disk failure.

Shared Memory Servers

Here the Servers Share Disks. Multiple servers simultaneously share access to disks. The main features are:

- Each computer has access to all of the volumes on all of the disks.
- This approach requires lock manager software to ensure that data can only be accessed by one computer at a time.
- Low network and server overhead.
- Reduced risk of downtime caused by disk failure.
- Usually used with disk mirroring or RAID technology.

Cluster Benefits

- Absolute scalability
- Incremental scalability
- High availability
- Superior price/performance

Cluster vs SMP

- Both provide multiprocessor support to high demand applications.
- Both available commercially
- SMP:
- Easier to manage and control
- Closer to single processor systems
- Scheduling is main difference
- Less physical space
- Lower power consumption
- Clustering:
- Superior incremental & absolute scalability
- Less cost
- Superior availability
- Redundancy

NONUNIFORM MEMORY ACCESS (NUMA) AND VECTOR

One another approach to multiprocessor organization is known as NonUniform Memory Access (NUMA). Some related definitions are:

a. Uniform memory access is where all processors have access to all parts of memory. Access time to all regions of memory is the same and access time to memory for different processors is the same(SMP).

b. Nonuniform memory access is where all processors have access to all parts of memory. Access time of processor differs depending on region of memory. Different processors access different regions of memory at different speeds.

c. Cache coherent NUMA(CC-NUMA) is where cache coherence is maintained among the caches of the various processors. It is significantly different from SMP and clusters.

NUMA is different from SMP and Clusters

- SMP has practical limit to number of processors
- Bus traffic limits to between 16 and 64 processors
- In clusters each node has own memory
- Apps do not see large global memory
- Coherence maintained by software not hardware
- NUMA retains SMP flavour while giving large scale multiprocessing.It's main objective is to maintain transparent system wide memory while permitting multiprocessor nodes, each with own bus or internal interconnection system.

CC-NUMA Organization

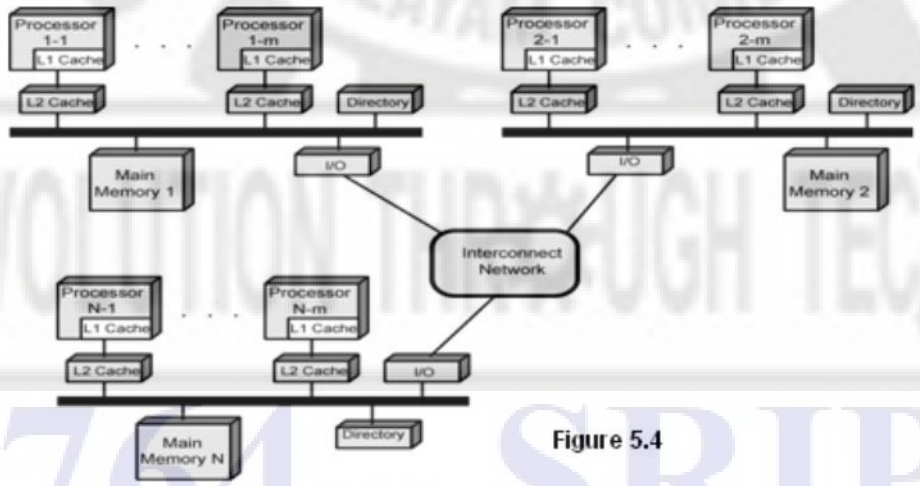


Figure 5.4

Figure 5.4 depicts a typical CC-NUMA organization. There are multiple independent nodes, each of which is an SMP organization. Thus, each node contains multiple processors, each with its own L1 and L2 caches, plus main memory. The node is the basic building block of the overall CC-NUMA organization. The nodes are interconnected by means of some communications facility, which could be a switching mechanism, a ring, or some other networking facility. Each node in the CC-NUMA system includes some main memory. From

the point of view of the processors, however, there is only a single addressable memory, with each location having a unique system wide address.

CC-NUMA Operation

When a processor initiates a memory access, if the requested memory location is not in that processor's cache, then the L2 cache initiates a fetch operation. If the desired line is in the local portion of the main memory, the line is fetched across the local bus. If the desired line is in a remote portion of the main memory, then an automatic request is sent out to fetch that line across the interconnection network, deliver it to the local bus, and then deliver it to the requesting cache on that bus. All of this activity is automatic and transparent to the processor and its cache

NUMA Pros & Cons

- Possibly effective performance at higher levels of parallelism than one SMP
- Not very supportive of software changes
- Performance can breakdown if too much access to remote memory
- Can be avoided by:
- L1 & L2 cache design reducing all memory access
- Need good temporal locality of software
- Not transparent
- Page allocation, process allocation and load balancing changes can be difficult.

Vector Computation

Introduction

Array processors are alternatives to supercomputer and are configured as peripherals to mainframe & minicomputers. They run vector portion of problems.

Figure 5.5 A Taxonomy of computer Organization

Vector Computation

The main reason to the design of a supercomputer or array processor is to recognize that the main task is to perform arithmetic operations on arrays or vectors of floating-point

numbers. In a general-purpose computer, this will require iteration through each element of the array. For example, consider two vectors (one-dimensional arrays) of 100 numbers, A and B .We would like to add these and place the result in C. This requires 100 separate additions in a conventional computer. The program coding is as follows:

```
      DO 20 I = 1, 100
20    C(I) = B(I) + A(I)
```

Conventional computer

```
      Initialize I = 0
20    Read A(I)
      Read B(I)
      Store C(I) = A(I) + B(I)
      Increment I = i + 1
      If I  100 goto 20
```

In the above program is a loop that reads a pair of operands from arrays A and B and performs a floating point addition. The loop control variable is updated and the steps are repeated 100 times.A computer capable of vector processing eliminates the overhead associated with time it takes to fetch and execute instructions in the loop. It allows operations to be specified with a single vector instruction of the form

Vector computer

```
      C(1:100) = A(1:100) + B(1:100)
```

The vector instruction includes the initial address of the operands, the length of the vectors and the operation to be performed, all in one composite instruction. General purpose computers rely on iteration to do vector calculations.

Approaches to Vector Computation

The preceding discussion describes approaches to vector computation in logical or architectural terms. Three main types of processor organization that can be used to implement these approaches are:

- Pipelined ALU
- Parallel ALUs
- Parallel processors

Figure 5.6a illustrates the first two of these approaches. Here the concept is extended to the operation of the ALU. Because floating-point operations are rather complex, there is opportunity for decomposing a floating-point operation into stages, so that different stages can operate on different sets of data concurrently. It should be clear that this organization is suitable for vector processing. The pipeline operation can be further enhanced if the vector elements are available in registers rather than from main memory. This is in fact suggested by Figure 5.6a. The elements of each vector operand are loaded as a block into a vector register, which is simply a large bank of identical registers. The result is also placed in a vector register. Thus, most operations involve only the use of registers, and only load and store operations at the beginning and end of a vector operation require access to memory.

Another way to achieve vector processing is by the use of multiple ALUs in a single processor, under the control of a single control unit. In this case, the control unit routes data to ALUs so that they can function in parallel. This is illustrated in Figure 5.3.2b .As with pipelined organization, a parallel ALU organization is suitable for vector processing.

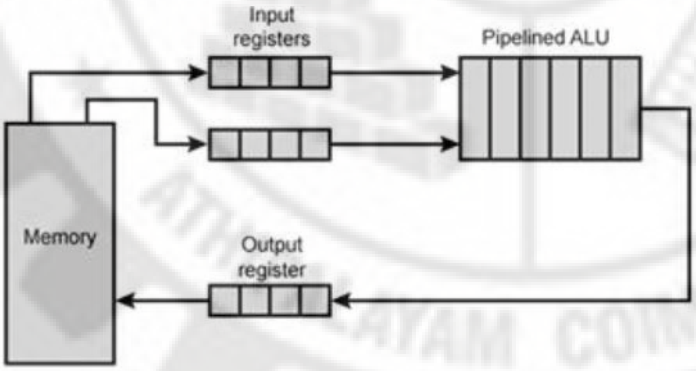


Figure 5.6 (a) Pipelined ALU

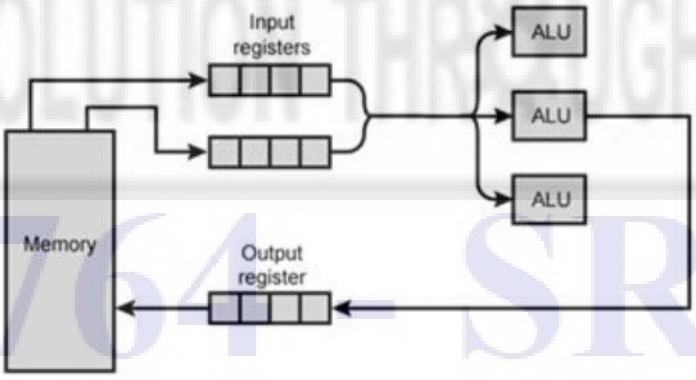


Figure 5.6 (b) Parallel ALUs

Finally, vector processing can be achieved by using multiple parallel processors. In this case, it is necessary to break the task up into multiple processes to be executed in parallel. This organization is effective only if the software and hardware for effective coordination of parallel processors is available.

The term vector processor is often equated with a pipelined ALU organization, although a parallel ALU organization is also designed for vector processing. A parallel processor organization may also be designed for vector processing. At present, the pipelined ALU organization dominates the marketplace. Pipelined systems are less complex than the other two approaches.

MULTICORE

A multicore computer, or chip multiprocessor, combines two or more processors on a single computer chip. Each core consists of all of the components of an independent processor, such as registers, ALU, pipeline hardware, and control unit, plus L1 instruction and data caches. In addition to the multiple cores, contemporary multicore chips also include L2 cache and, in some cases, L3 cache.

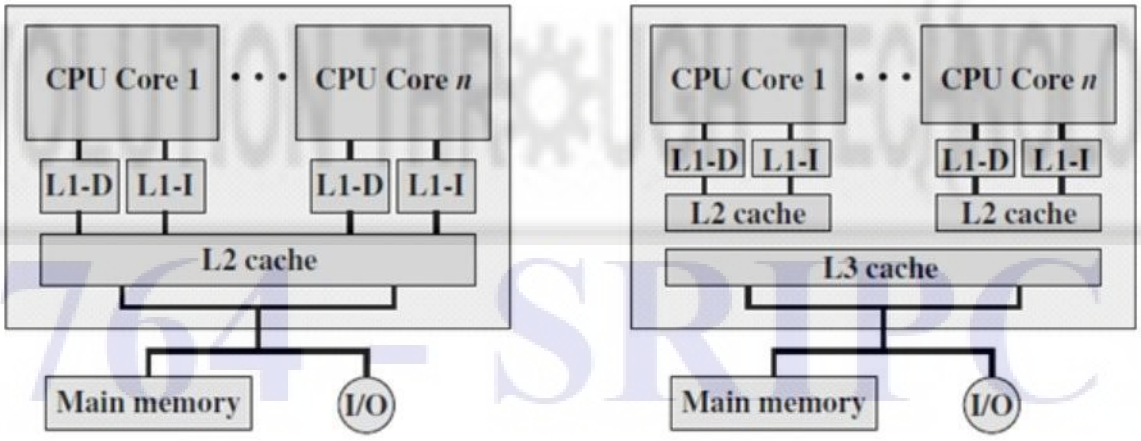
The main variables in a multicore organization are as follows:

- The number of core processors on the chip
- The number of levels of cache memory
- The amount of cache memory that is shared Module.

There are four general organizations for multicore systems:

1.Shared L2 cache
2. Shared L3 cache
3.Dedicated L1 cache
4.Dedicated L2 cache

Figure 5.7 shows the above 4 organization of multicore systems.



(c) Shared L2 cache (d) Shared L3 cache

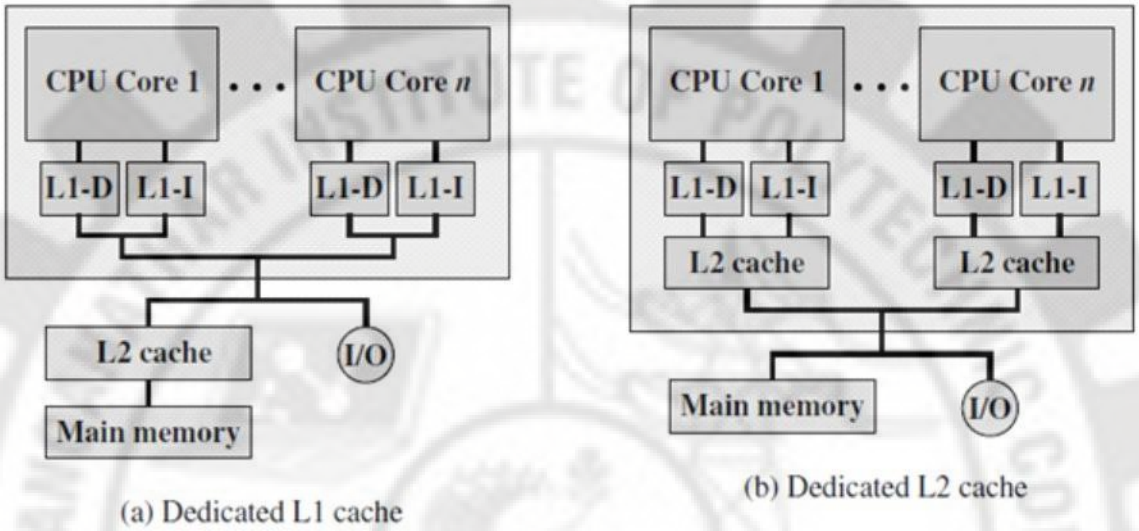


Figure 5.7

Figure 5.7a is an organization in which each core has its own dedicated L1 cache. Almost invariably, the L1 cache is divided into instruction and data caches. The organization of Figure 5.7b is one in which there is no on-chip cache sharing. In this, there is enough area available on the chip to allow for L2 cache. Figure 5.7c shows a similar allocation of chip space to memory, but with the use of a shared L2 cache. Figure 5.7d, shows a shared L3 cache , with dedicated L1 and L2 caches for each core processor. The Intel Core i7 is an example of this organization.

The use of a shared L2 cache on the chip has several advantages over dedicated caches:

- Constructive interference can reduce overall miss rates.
- A related advantage is that data shared by multiple cores is not replicated at the shared cache level.
- With proper frame replacement algorithms, the amount of shared cache allocated to
- each core is dynamic, so that threads that have a less locality can employ more cache. Interprocessor communication is easy to implement, via shared memory locations.
- The use of a shared L2 cache confines the cache coherency problem to the L1 cache level.

A potential advantage to having only dedicated L2 caches on the chip is that each core enjoys more rapid access to its private L2 cache. This is advantageous for threads that exhibit strong locality. As both the amount of memory available and the number of cores grow, the use of a shared L3 cache combined with either a shared L2 cache or dedicated per core L2 caches seems likely to provide better performance than simply a massive shared L2 cache.

Intel x86 Multicore Organization

Intel Core Duo uses superscalar cores and Intel Core i7 uses simultaneous multi-threading (SMT).

Core2 Duo

The Intel Core Duo, introduced in 2006, implements two x86 superscalar processors with a shared L2 cache .The general structure of the Intel Core Duo is shown in Figure 5.8. Main features are

- Two x86 superscalar, shared L2 cache
- Dedicated L1 cache per core
- 32KB instruction and 32KB data
- Thermal control unit per core
- Manages chip heat dissipation
- Maximize performance within constraints
- Improved ergonomics
- Advanced Programmable Interrupt Controlled (APIC)
- Inter-process interrupts between cores
- Routes interrupts to appropriate core
- Includes timer so OS can interrupt core
- Power Management Logic
- Monitors thermal conditions and CPU activity
- Adjusts voltage and power consumption
- Can switch individual logic subsystems
- 2MB shared L2 cache
- Dynamic allocation
- MESI support for L1 caches
- Extended to support multiple Core Duo in SMP
- L2 data shared between local cores or external
- Bus interface

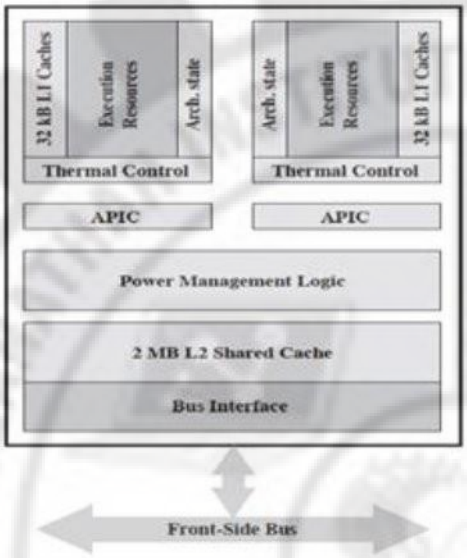


Figure 5.8 INTEL Core Duo Block Diagram

Core i7

The Intel Core i7, introduced in November of 2008, implements four x86 SMT processors, each with a dedicated L2 cache, and with a shared L3 cache. The general structure of the Intel Core i7 is shown in Figure 5.9. Each core has its own dedicated L2 cache and the four cores share an 8-MB L3 cache. The main features are:

- Four x86 SMT processors
- Dedicated L2, shared L3 cache
- Speculative pre-fetch for caches
- On chip DDR3 memory controller
- Three 8 byte channels (192 bits) giving 32GB/s
- No front side bus
- QuickPath Interconnection
- Cache coherent point-to-point link
- High speed communications between processor chips
- 6.4G transfers per second, 16 bits per transfer
- Dedicated bi-directional pairs
- Total bandwidth 25.6GB/s

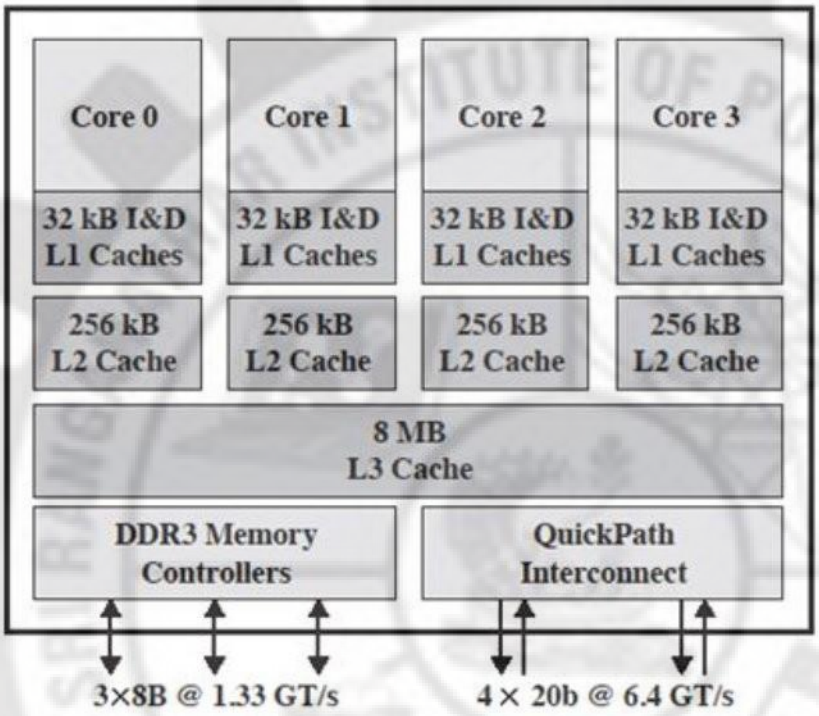


Figure 5.9 Intel Core i7 Block Diagram

Summary

- In a SMP, multiple processors share a single memory or pool of memory by means of a shared bus or other interconnection mechanism. Availability, Incremental Growth and Scaling are some of the features of an SMP.
- Multithreading is the ability of a central processing unit (CPU) or a single core in a multi-core processor to execute multiple processes or threads concurrently. Different approaches to explicit multithreading is possible.
- Clusters are collection of independent whole uniprocessors or SMPs usually called nodes interconnected to form a cluster. There are different types of clusters viz., Shared disk clusters, Shared memory clusters and Separate server clusters.
- Nonuniform memory access is where all processors have access to all parts of memory. A NUMA system without cache coherence is more or less equivalent to a cluster. The commercial products that have received much attention recently are CC-NUMA systems, which are quite distinct from both SMPs and clusters.
- Supercomputers were developed to handle mathematical problems of physical processes.These machines are typically capable of billions of floating-point operations per second. There is another type of system that has been designed to address the need for vector computation, referred to as the array processor. The term vector processor is often equated with a pipelined ALU organization, although a

parallel ALU organization is also designed for vector processing. At present, the pipelined ALU organization dominates the marketplace.

- Multicore processors are designed to adhere to reasonable power consumption, heat dissipation, and cache coherence protocols. In order to use a multicore processor at full capacity the applications running on the system must be multithreaded. There are four general multicore organizations. Core2 Duo and Core i7 are examples of multicore processor organization discussed in this unit.

REVIEW QUESTIONS

Two mark questions

1. What is an SMP.
2. Define Symmetric Multiprocessing.
3. List any 2 characteristics of a SMP.
4. List the features of a bus organization.
5. What is the drawback of a bus organization.
6. What is a thread.
7. What is a process.
8. What is multithreading.
9. What is a cluster.
10. Distinguish a user level thread from a kernel level thread.
11. What is a process switch.
12. What is a thread switch.
13. What is Uniform Memory Access.
14. What is NUMA.
15. What is CC-NUMA.
16. What is vector computation.
17. What are the various types of processors that are suitable for Vector Computation.
18. What is a multicore computer or a chip multiprocessor.
19. What are the main variables in a multicore organization.
20. List down the four general organizations of a multicore processor.
21. List any two advantages of using a dedicated L2 cache over a shared L2 cache.

Three mark questions

1. What are the various characteristics of a SMP.
2. List down the advantages of a SMP organization over a Uniprocessor organization.
3. How is time sharing achieved in an SMP.
4. Explain the features of an SMP.
5. What are the different types of threads available.
6. List down the characteristics of a process.
7. Distinguish a thread from a Process.
8. What are the benefits of clustering.
9. Differentiate NUMA from SMP and Clusters.
10. List the Pros and Cons of NUMA.
11. What are the various approaches to vector computation.
12. What are the advantages of using dedicated caches over shared caches.

Five / Ten mark questions

1. Explain the organization of a multiprocessor system in detail with a neat block diagram.

2. Explain in detail the organization of a Z990 main frame.
3. What are the various approaches to explicit multithreading.
4. What are the various cluster configurations available. Explain with block diagram.
5. How can you classify the clusters. List down their features.
6. Distinguish a SMP from a Cluster.
7. Explain CC-NUMA organization and operation with block diagram.
8. Distinguish between vector computation using a general computer and a vector processor with example.
9. Explain the various approaches to vector computation with neat block diagram.
10. Explain with block diagram the four general organizations of a multicore system.
11. Explain the organization and features of a Core2 Duo processor.
12. Explain the organization and features of a Core i7 processor.