Introduction

- A general purpose computer should have the ability to exchange information with a wide range of devices in varying environments.
- Computers can communicate with other computers over the Internet and access information around the globe.
- They are an integral part of home appliances, manufacturing equipment, transportation systems, banking and point-of-sale terminals.

Accessing I/O Devices

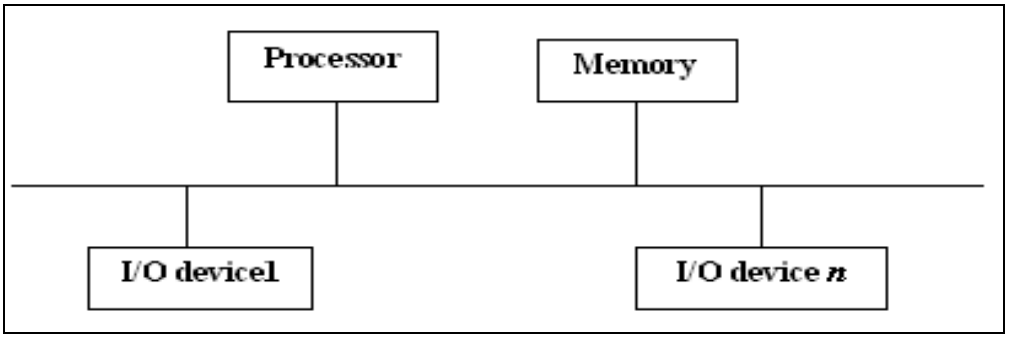


Figure: A single-bus structure

The I/O devices are connected by using a single bus which enables data transaction between each device as shown in figure.

- Bus consists of 3 sets of lines used to carry address, data and control signals.
- Each I/O device is assigned a unique set of address. When the processor places a particular address on the address lines, the device that recognizes this address responds to the commands issued on the control lines.
- The processor requests either a read or a write operation which is transferred over the data lines. When I/O devices and the memory share the same address space, the arrangement is called memory-mapped I/O. There are two ways to deal with I/O devices as shown above figure.

Memory mapped I/O

The Memory and I/O devices are sharing information by a common address-space. Any data-transfer instruction (like Move, Load) can be used to exchange information. For example,

Move DATAIN, R0;

- This instruction reads data from DATAIN (address of input-buffer associated with Keyboard) & stores them into processor-register R0.

Move R0, DATAOUT

- This instruction sends the contents of register R0 to location DATAOUT, which may be the output data buffer of a display unit or a printer.

I/O mapped I/O

In I/O mapped I/O, the memory and I/0 address-spaces are different. Special instructions are used for data transfer such as IN and OUT.

- The I/O devices use special I/O address space or memory address space.
- The I/O devices examine the low-order bits of the address bus to determine whether they should respond.
- Advantage of separate I/O space: I/O devices deal with fewer address-lines.

The hardware required to connect an I/O device to the bus as shown in the figure.

- The address decoder enables the device to recognize its address when this address appears on the address lines.
- The data register holds the data being transferred to or from the processor.
- The status register contains information relevant to the operation of the I/O device.
- Both the data and status registers are connected to the data bus and assigned unique addresses. The address decoder, the data and status registers, and the control circuitry required to coordinate I/O transfers constitute **the device's interface circuit.**

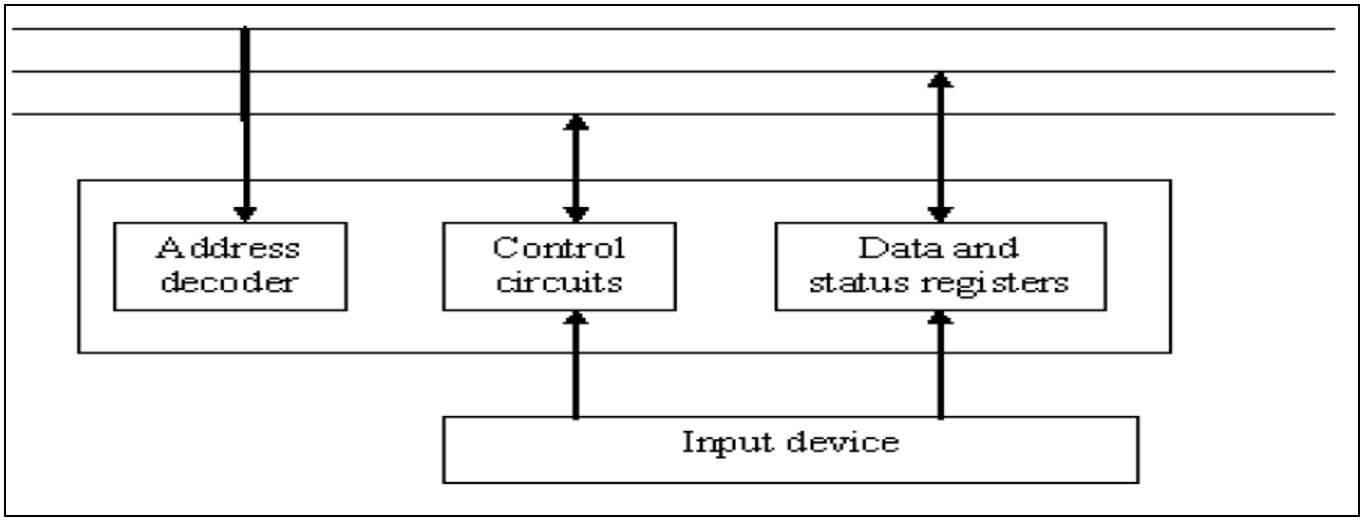


Figure: I/O interface for an input device

Program Controlled I/O

- Processor repeatedly checks a status-flag to achieve required synchronization between processor & input/output device. (We say that the processor polls the device).

Main drawback: The processor wastes its time in checking the status of the device before actual data transfer takes place.

Mechanisms Used For Interfacing I/O Operations:

Interrupt I/O

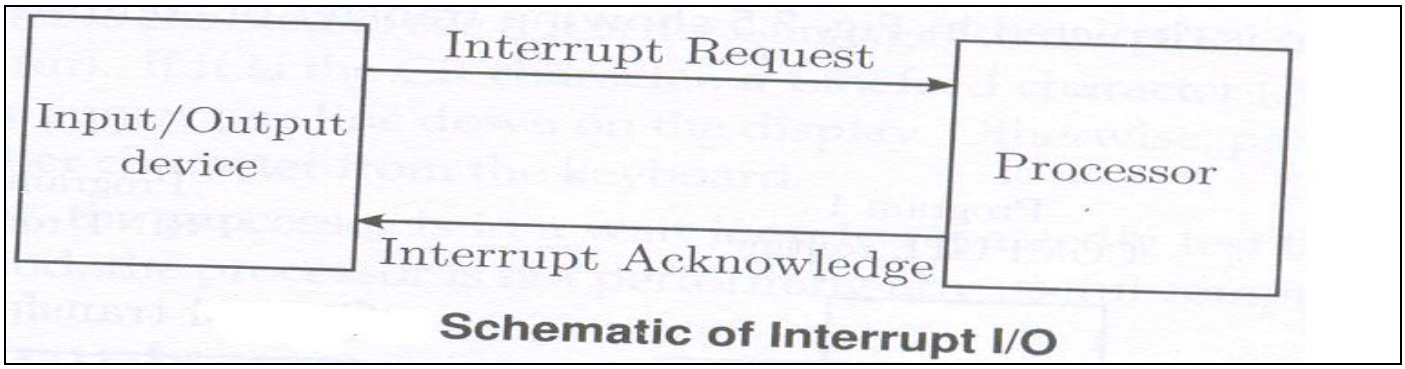
- Synchronization is achieved by having I/O device send a special signal over bus whenever it is ready for a data transfer operation.

Direct Memory Access (DMA)

- This involves having the device-interface transfer data directly to or from the memory without continuous involvement by the processor. This technique is used for high-speed I/O devices.

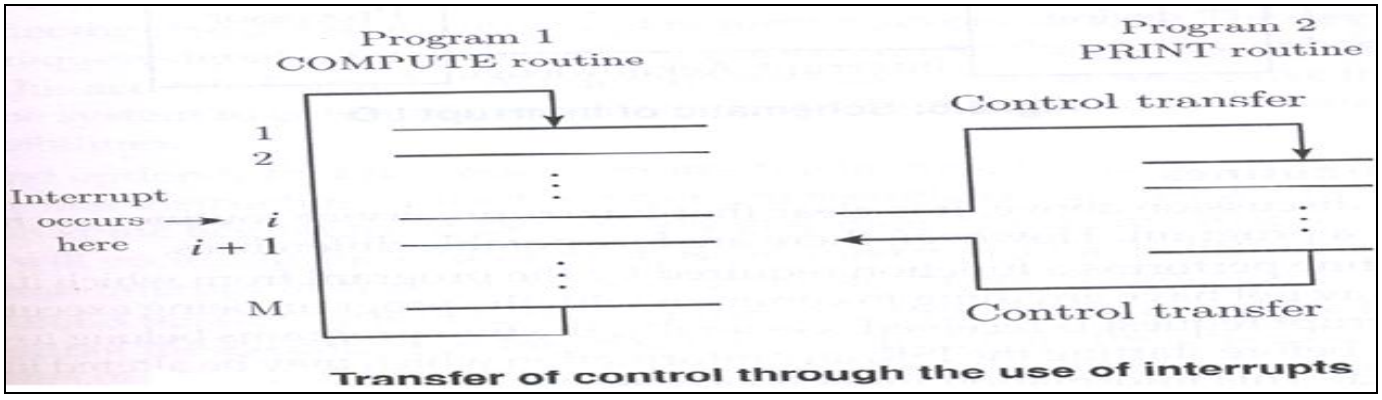
Interrupts

- There are many situations where other tasks can be performed while waiting for an I/O device to become ready. A hardware signal called an Interrupt will alert the processor when an I/O device becomes ready. It can do so by sending a hardware signal **called an interrupt to the processor.**



Schematic of Interrupt I/O

- For example, consider, COMPUTE and PRINT routines. The routine executed in response to an interrupt request is **called interrupt-service routine**.
- Transfer of control through the use of interrupts happens. The processor must inform the device that its request has been recognized by sending interrupt-acknowledge signal.



Transfer of control through the use of interrupts

- One must therefore know the difference between **Interrupt Vs Subroutine**. Interrupt latency is concerned with saving information in registers will increase the delay between the time an interrupt request is received and the start of execution of the interrupt-service routine.
- Processor is executing the instruction located at address i when an interrupt occurs.
- Routine executed in response to an interrupt request is called the interrupt-service routine.
- When an interrupt occurs, control must be transferred to the interrupt service routine.
- But before transferring control, the current contents of the PC (i+1), must be saved in a known location.
- This will enable the return-from-interrupt instruction to resume execution at i+1.
- Return address, or the contents of the PC are usually stored on the processor stack.

Interrupt latency: *Delay between the time an interrupt request is received and the start of execution of the interrupt service routine.*

Interrupt Hardware:-

- An I/O device requests an interrupt by activating a bus-line called **interrupt-request (IR).**
- A single interrupt request line may be used to serve n devices as depicted. All devices are connected to the line via switches to ground. To request an interrupt, a device closes its associated switch. Thus, if all interrupt-request signals $INTR_1$ to INTRn are inactive, that is, if all switches are open, the voltage on the interrupt-request line will be equal to $V_{dd.}$
- This is the inactive state of the line. Since the closing of one or more switches will cause the line voltage to drop to 0, causing the interrupt request signal , INTR received by the processor to go to 1.
- The value of INTR is the logical OR of the requests from individual devices, that is,

$INTR = INTR_1 + \ldots\ldots\ldots + INTRn$

It is customary to use the complemented form, INTR, to name the interrupt-request signal on the common line, because this signal is active when in the low-voltage state.

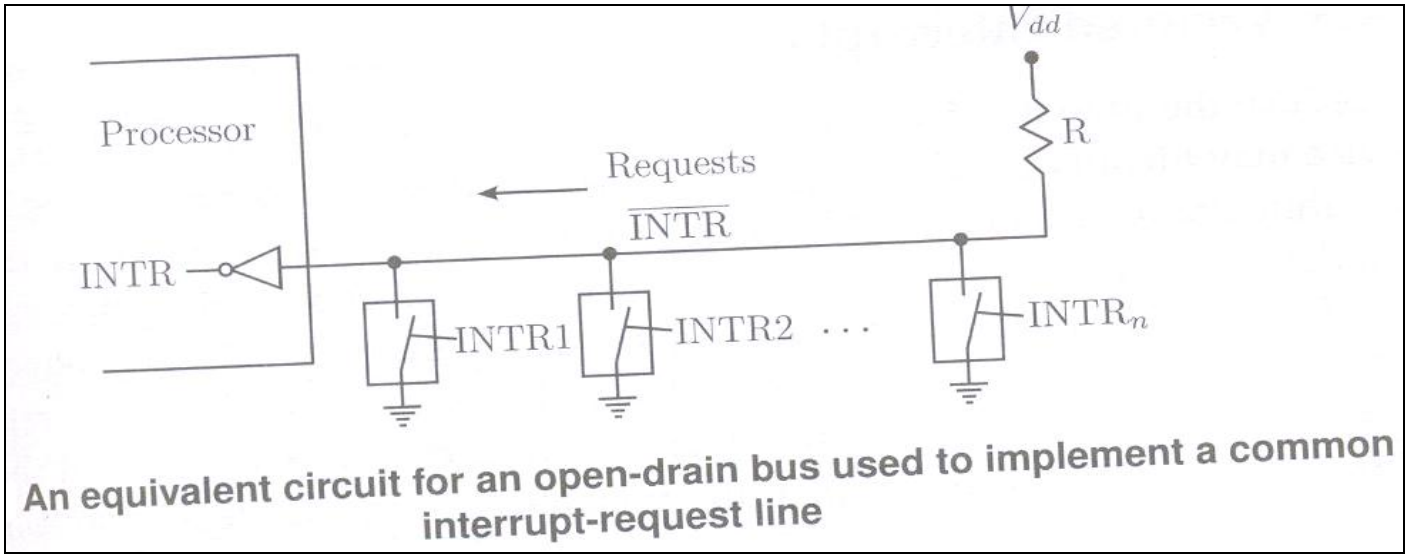


Figure: An equivalent circuit for an open-drain bus used to implement a common interrupt-request line

- The figure shows that special gates known as open-collector (for bipolar circuits) or open-drain (for MOS circuits) are used to drive the INTR line. The output of an open-collector or an open-drain gate is equivalent to a switch to ground that is open when the gate's input is in the 0 state and closed when it is in the 1 state. Resistor R is called a pull-up resistor because it pulls the line voltage up to the high-voltage state when the switches are open.

Enabling and Disabling Interrupts

- The facilities provided in a computer must give the programmer complete control over the events that take place during program execution. The arrival of an interrupt request from an external

device causes the processor to suspend the execution of one program and start the execution of another.

- Because interrupts can arrive at any time, they may alter the sequence of events from the envisaged by the programmer. Hence, the interruption of program execution must be carefully controlled.
- Let us consider in detail the specific case of a single interrupt request from one device. When a device activates the interrupt-request signal, it keeps this signal activated until it learns that the processor has accepted its request.
- This means that the interrupt-request signal will be active during execution of the interrupt-service routine, perhaps until an instruction is reached that accesses the device .
- This activated signal, if not deactivated, may lead to successive interruptions, causing the system to enter into an infinite loop.

To prevent the system from entering into an infinite-loop because of interrupt, there are 3 possibilities:

First Option

- The first possibility is to have the processor hardware ignore the interrupt-request line until the execution of the first instruction of the interrupt-service routine has been completed.
- Then, by using an Interrupt-disable instruction as the first instruction in the interrupt-service routine, the programmer can ensure that no further interruptions will occur until an Interrupt-enable instruction is executed.
- Typically, the Interrupt-enable instruction will be the last instruction in the interrupt-service routine before the Return-from-interrupt instruction. The processor must guarantee that execution of the Return-from-interrupt instruction is completed before further interruption can occur.

Second Option

- The second option is to have the processor automatically disable interrupts before starting the execution of the ISR.
- After saving the contents of the PC and the processor status register (PS) on the stack, the processor performs the equivalent of executing an Interrupt-disable instruction. It is often the case that one bit in the PS register, called **Interrupt-enable,** indicates whether interrupts are enabled.

Third option

- In the third option, the processor has a special interrupt-request line for which the interrupt-handling circuit responds only to the leading edge of the signal. Such a line is said to be edge-triggered. Processor will receive only one request hence there will not be multiple interruption.

Summary:

Sequence of events involved in handling an interrupt-request from a single device is as follows:

1. The device raises an interrupt request.
2. The processor interrupts the program currently being executed.
3. Interrupts are disabled by changing the control bits in the PS (except in the case of edge-triggered interrupts).
4. The device is informed that its request has been recognized, and in response, it deactivates the interrupt-request signal.

5. The action requested by the interrupt is performed by the interrupt-service routine.
6. Interrupts are enabled and execution of the interrupted program is resumed.

Handling Multiple Devices

- An I/O device requests a service by activating a bus line called interrupt request. There may be more than one I/O devices in a system which request service. Since these devices operate independently, there is no specific order in which they generate interrupts. In another case, several devices may send request at the same time.

.

When several devices requests interrupt at the same time, it raises some questions. They are

- How can the processor recognize the device requesting an interrupt?
- How can the processor obtain the starting address of the appropriate routine?
- Should a device be allowed to interrupt the processor while another interrupt is being serviced?
- How should two or more simultaneous interrupt requests be handled?
- Information needed to determine whether a device is requesting an interrupt is available in its status-register. When a device raises an interrupt-request, it sets IRQ bit to 1 in its status-register.
- KIRQ and DIRQ are the interrupt-request bits for keyboard & display. Simplest way to identify interrupting device is to have ISR poll all I/O devices connected to bus.
- The first device encountered with its IRQ bit set is the device that should be serviced. After servicing this device, next requests may be serviced.

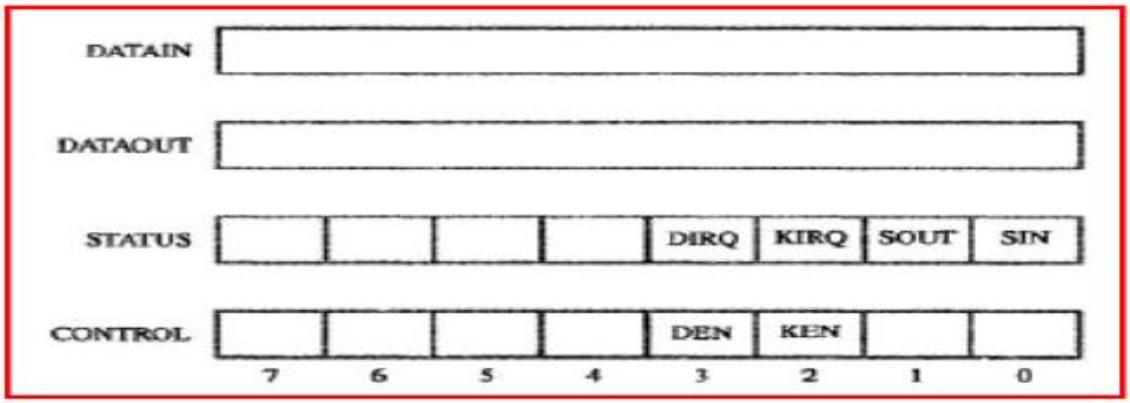


Figure 5: Registers in keyboard and display interfaces

```
        Move        #LINE,R0          Initialize memory pointer.
WAITK   TestBit     #0,STATUS         Test SIN.
        Branch=0    WAITK             Wait for character to be entered.
        Move        DATAIN,R1         Read character.
WAITD   TestBit     #1,STATUS         Test SOUT.
        Branch=0    WAITD             Wait for display to become ready.
        Move        R1,DATAOUT        Send character to display.
        Move        R1,(R0)+          Store charater and advance pointer.
        Compare     #$0D,R1           Check if Carriage Return.
        Branch≠0    WAITK             If not, get another character.
        Move        #$0A,DATAOUT      Otherwise, send Line Feed.
        Call        PROCESS           Call a subroutine to process the
                                        the input line.
```

Figure 6: A program that reads one line from the keyboard, stores it in memory buffer, and echoes it back to the display

Vectored Interrupts

- To reduce the polling process time, a device requesting an interrupt may identify itself directly to the processor. Now, the processor can immediately start executing the corresponding ISR. Such interrupts are called as **vectored interrupts**. A device requesting an interrupt can identify itself by sending a special code to the processor over the bus.
- This enables the processor to identify individual devices even if they share a single interrupt-request line. The code supplied by the device may represent the starting address of the interrupt-service routine for that device.
- The code length is typically in the range of 4 to 8 bits. The remainder of the address is supplied by the processor based on the area in its memory where the addresses for interrupt-service routines are located.
- The address stored at the location pointed to by interrupting-device is called the interrupt-vector. The processor reads this address, called the *interrupt vector,* and loads it into PC then executes appropriate ISR. Interrupting-device must wait to put data on bus only when processor is ready to receive it.
- When processor is ready to receive interrupt-vector code, it activates INTA line. I/O device responds by sending its interrupt-vector code & turning off the INTR signal.

Interrupt Nesting

- Computer keeps track of the time of the day using real-time clock. Device sends IR to processor at regular intervals to update time in seconds, minutes.
- For some device, a long delay is responding to an interrupt request may cause error in the operation of computer. Such interrupts are acknowledge and serviced even though processor is executing an interrupt service routine for another device system of interrupts that allows an interrupt service routine to be implemented is known as **nested interrupts**.

Multiple Priority Scheme:

- In multiple level priority schemes, we assign a priority level to the processor that can be changed under program control.
- The priority level of the processor is the priority of the program that is currently being executed.
- The processor accepts interrupts only from devices that have priorities higher than its own.
- At the time of execution of an ISR for some device is started, the priority of the processor is raised to that of the device.
- The action disables interrupts from devices at the same level of priority or lower.

Privileged Instruction:

- The processor priority is usually encoded in a few bits of the Processor Status word. It can also be changed by program instruction & then it is writing into PS. These instructions are called **privileged instruction**.
- This can be executed only when the processor is in supervisor mode. The processor is in supervisor mode only when executing OS routines. It switches to the user mode before beginning to execute application program.

Privileged Exception:

- Processor will have privileged instructions to run in supervisor mode.

- User program cannot accidently or intentionally change the priority of the processor & disrupts the system operation. An attempt to execute a privileged instruction while in user mode, leads to a special type of interrupt called the **privileged exception.**
- Each of the interrupt request line is assigned a different priority level. Interrupt request received over these lines are sent to a priority arbitration circuit in the processor. A request is accepted only if it has a higher priority level than that currently assigned to the processor

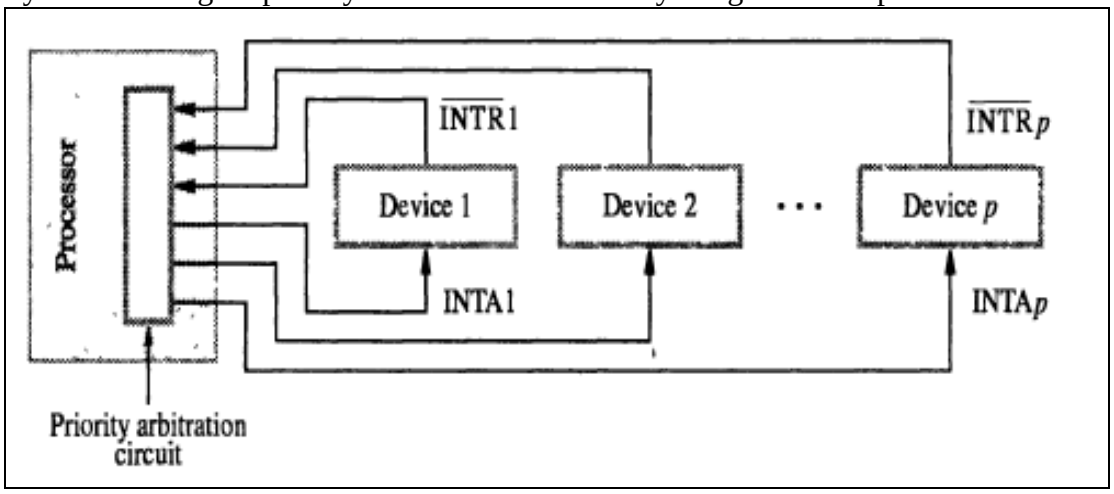


Figure 7: implementation of interrupt priority using individual interrupt-request and acknowledge lines

Simultaneous Requests:

Daisy Chain:

- The interrupt request line INTR is common to all devices. The interrupt acknowledge line INTA is connected in a daisy chain fashion such that INTA signal propagates serially through the devices.

- When several devices raise an interrupt request, the INTR is activated & the processor responds by setting INTA line to 1. This signal is received by device. Device1 passes the signal on to device2 only if it does not require any service. If devices1 has a pending request for interrupt blocks that INTA signal & proceeds to put its identification code on the data lines. Therefore, the device that is electrically closest to the processor has the highest priority.

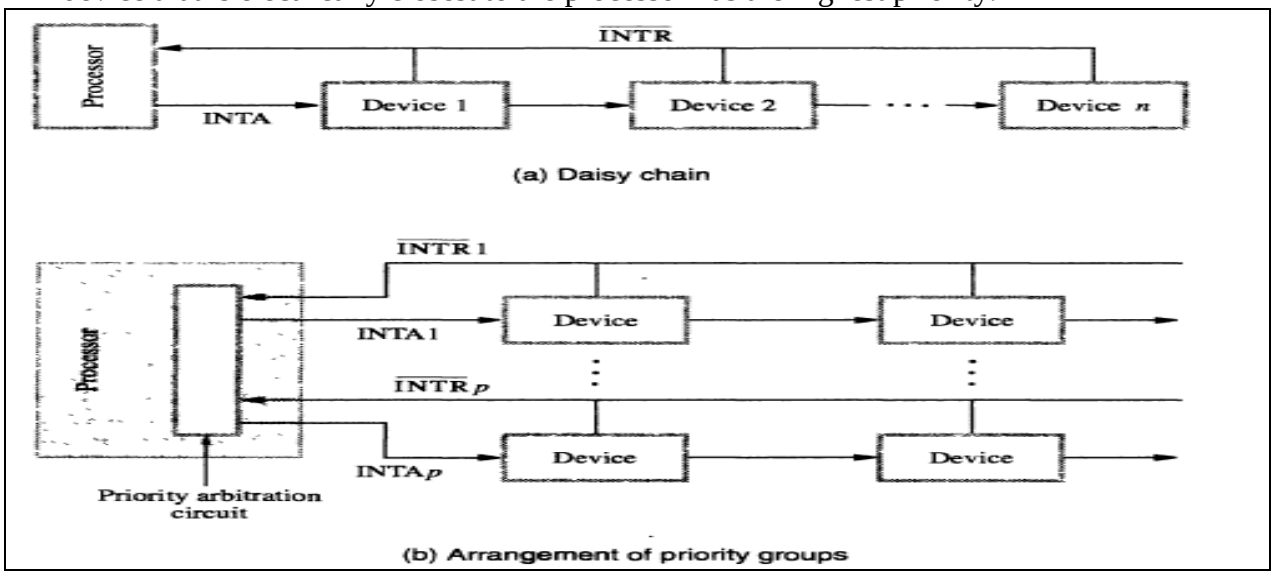


Figure 8: Interrupt priority schemes

Arrangement of Priority Groups*:*

Here the devices are organized in groups & each group is connected at a different priority level. Within a group, devices are connected in a daisy chain.

Controlling Device Requests

- The control needed is usually provided in the form of an interrupt-enable bit in the device's interface circuit.
- KEN = Keyboard Interrupt Enable
- DEN = Display Interrupt Enable
- KIRQ / DIRQ = Keyboard / Display unit requesting an interrupt.

There are two mechanisms for controlling interrupt requests.

- ***At the devices end***, an interrupt enable bit in a control register determines whether the device is allowed to generate an interrupt requests.
- ***At the processor end***, either an interrupt enable bit in the PS (Processor Status) or a priority structure determines whether a given interrupt requests will be accepted.

Initiating the Interrupt Process:

1. Load the starting address of ISR in location INTVEC (vectored interrupt).
2. Load the address LINE in a memory location PNTR. The ISR will use this location as a pointer to store the input characters in the memory.
3. Enable the keyboard interrupts by setting bit 2 in register CONTROL to 1.
4. Enable interrupts in the processor by setting to 1, the IE bit in the processor status register PS.

Exception of ISR:

1. Read the input characters from the keyboard input data register. This will cause the interface circuits to remove its interrupt requests.
2. Store the characters in a memory location pointed to by PNTR & increment PNTR.
3. When the end of line is reached, disable keyboard interrupt & inform program main.
4. Return from interrupt.

Exceptions

An interrupt is an event that causes the execution of one program to be suspended and the execution of another program to begin. The Exception is used to refer to any event that causes an interruption.

Recovery from Errors:

- Computers have error-checking code in Main Memory, which allows detection of errors in the stored data. If an error occurs, the control hardware detects it informs the processor by raising an interrupt.
- The processor also interrupts the program, if it detects an error or an unusual condition while executing the instance (i.e.) it suspends the program being executed and starts an execution service routine. This routine takes appropriate action to recover from the error.

Debugging:
System software has a program called debugger, which helps to find errors in a program. The debugger uses exceptions to provide two important facilities. They are Trace and ***Breakpoint.***

Trace Mode:

- When processor is in trace mode, an exception occurs after execution of every instance using the debugging program as the exception service routine. The debugging program examine the contents of registers, memory location etc.
- On return from the debugging program the next instance in the program being debugged is executed. The trace exception is disabled during the execution of the debugging program.

Break point:

- Here the program being debugged is interrupted only at specific points selected by the user. An instance called the Trap (or) software interrupt is usually provided for this purpose.
- While debugging the user may interrupt the program execution after instance 'I'. When the program is executed and reaches that point it examine the memory and register contents.

Privileged Exception:

- To protect the OS of a computer from being corrupted by user program certain instance can be executed only when the processor is in supervisor mode.
- These are called privileged exceptions. When the processor is in user mode, it will not execute instance (i.e.) when the processor is in supervisor mode, it will execute instance.

Direct Memory Access (DMA)

- A special control unit may be provided to allow the transfer of large block of data at high speed directly between the external device and main memory, without continuous intervention by the processor. This approach is called direct memory access (**DMA)**. DMA transfers are performed by a control circuit called the **DMA Controller**.

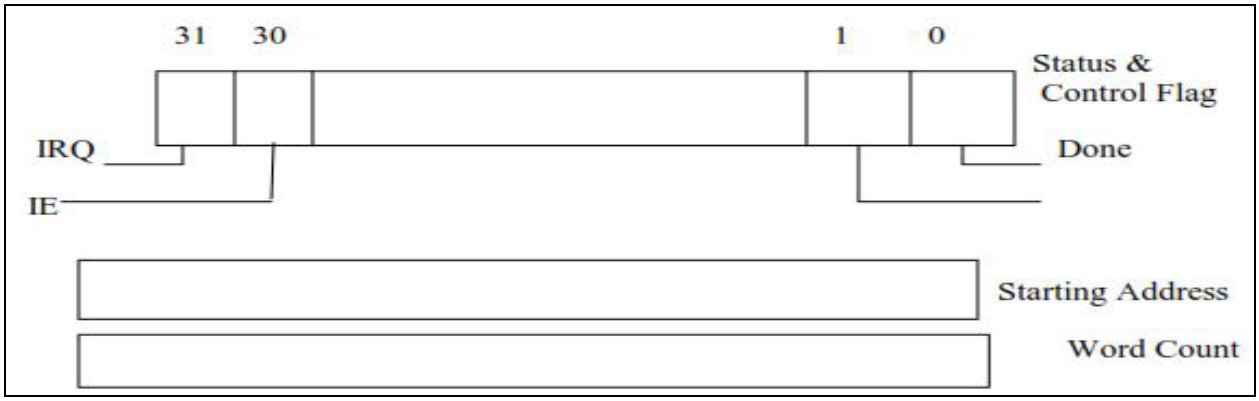


Figure 9: Registers in a DMA Interface

- To initiate the transfer of a block of words, the processor sends, are ***Starting address, Number of words in the block and Direction of transfer.***
- When a block of data is transferred, the DMA controller increment the memory address for successive words and keep track of number of words and it also informs the processor by raising an interrupt signal.

- While DMA control is taking place, the program requested the transfer cannot continue and the processor can be used to execute another program. After DMA transfer is completed, the processor returns to the program that requested the transfer.

The above figure shows an example of the DMA controller registers that are accessed by the processor to initiate transfer operations. Two registers are used for storing the Starting address and the word count. The third register contains status and control flags. The R/W bit determines the direction of the transfer. When

- R/W =1, DMA controller read data from memory to I/O device.
- R/W =0, DMA controller perform write operation.
- Done Flag=1, the controller has completed transferring a block of data and is ready to receive another command.
- IE=1, it causes the controller to raise an interrupt (interrupt Enabled) after it has completed transferring the block of data.
- IRQ=1, it indicates that the controller has requested an interrupt.

A DMA controller connects a high speed network to the computer bus. The disk controller two disks, also has DMA capability and it provides two DMA channels. To start a DMA transfer of a block of data from main memory to one of the disks, the program write s the address and the word count information into the registers of the corresponding channel of the disk controller. When DMA transfer is completed, it will be recorded in status and control registers of the DMA channel (i.e.) done bit=IRQ=IE=1.

Cycle Stealing:

- Memory accesses by the processor and the DMA controller are interwoven. Requests by DMA devices for using the bus are always given higher priority than processor requests.
- Among different DMA devices, top priority is given to high-speed peripherals such as a disk, a high-speed network interface, or a graphics display device.
- Since the processor originates most memory access cycles, the DMA controller can be said to **"steal" memory** cycles from the processor. Hence, the interweaving technique is usually called **cycle stealing.**

Burst Mode:

- The DMA controller may be given exclusive access to the main memory to transfer a block of data without interruption. This is known as Burst/Block Mode.

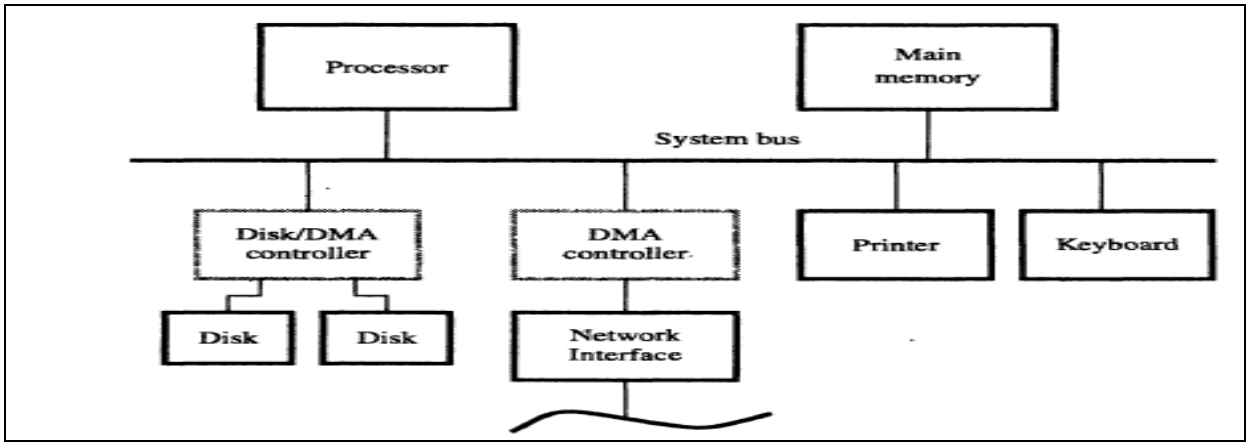


Figure 10: Use of DMA controllers in a computer system

Bus Arbitration

Bus Master:
The device that is allowed to initiate data transfers on the bus at any given time is called the **bus master**.

Bus Arbitration:

It is the process by which the next device to become the bus master is selected and the bus mastership is transferred to it.

There are 2 approaches to bus arbitration. They are,

1. Centralized arbitration (A single bus arbiter performs arbitration)
2. Distributed arbitration (all devices participate in the selection of next bus master).

Centralized Arbitration:

- Here the processor is the bus master and it may grants bus mastership to one of its DMA controller. A DMA controller indicates that it needs to become the bus master by activating the Bus Request line (BR) which is an open drain line. The signal on BR is the logical OR of the bus request from all devices connected to it.
- When BR is activated the processor activates the Bus Grant Signal (BGI) and indicated the DMA controller that they may use the bus when it becomes free. This signal is connected to all devices using a daisy chain arrangement. If DMA requests the bus, it blocks the propagation of Grant Signal to other devices and it indicates to all devices that it is using the bus by activating open collector line, Bus Busy (BBSY).

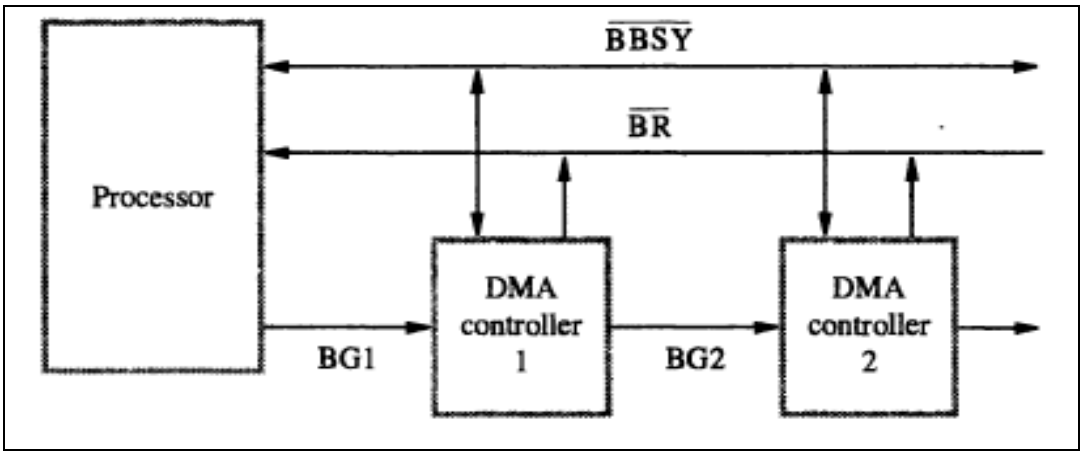


Figure 11: A simple arrangement for bus arbitration using a daisy chain

The timing diagram shows the sequence of events for the devices connected to the processor is shown. DMA Controller 2 requests and acquires bus mastership and later releases the bus. During its tenure as bus master, it may perform one or more data transfer operations, depending on whether it is operating in the cycle stealing or block mode. After it releases the bus, the processor resumes bus mastership.

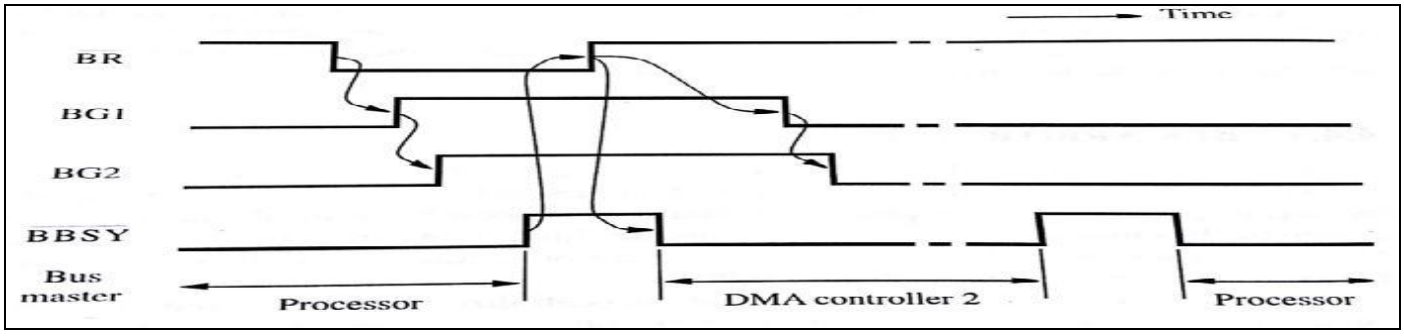


Figure 12: Sequence of signals during transfer of bus mastership for the devices in figure 11.

Distributed Arbitration:

- Distributed arbitration means that all devices waiting to use the bus have equal responsibility in carrying out the arbitration process, without using a central arbiter.
- Each device on the bus is assigned a 4 bit id. When one or more devices request the bus, they assert the Start-Arbitration signal and place their 4 bit ID number on four open collector lines, ARB0 to ARB3.
- A winner is selected as a result of the interaction among the signals transmitted over these lines. The net outcome is that the code on the four lines represents the request that has the highest ID number. The drivers are of open collector type.
- Hence, if the input to one driver is equal to 1, the input to another driver connected to the same bus line is equal to "0" the bus will be in low-voltage state.

Example:

- Assume two devices A and B have their ID 5 (0101), 6(0110) and their code is 0111. Each device compares the pattern on the arbitration line to its own ID, starting from most significant bit (MSB).
- If it detects a difference at any bit position, it disables the drivers at that bit position and for all lower-order bits. It does this by placing "0" at the input of these drivers. In our example, device "A" detects a difference in line ARB1.

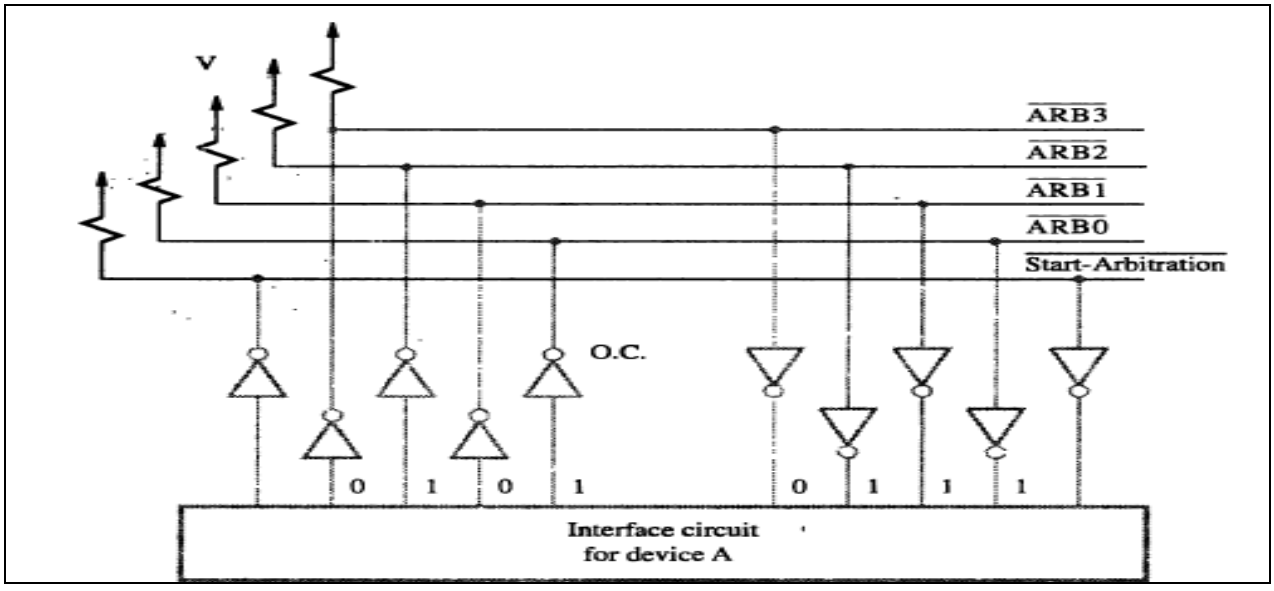


Figure 13: A distributed arbitration scheme

- Hence it disables the drivers on lines ARB1 and ARB0. This causes the pattern on the arbitration line to change to 0110 which means that "B" has won the contention.
- It has the advantage of offering higher reliability, because operation of the bus is not dependent on any single device.

Buses

- A bus protocol is the set of rules that govern the behavior of various devices connected to the bus as to when to place information on the bus, assert control signals, and so on. The bus lines used for transferring data is grouped into 3 types.
- They are Address line, Data line and Control line.
- Control signals Specifies that whether read or write operation has to perform and also carries timing information. They specify the time at which the processor & I/O devices place the data on the bus and receive the data from the bus.
- During data transfer operation, one device plays the role of a "Master". Master device initiates the data transfer by issuing read or write command on the bus. Hence it is also called as "Initiator". The device addressed by the master is called as Slave or Target.

Types of Buses:

There are 2 types of buses. They are Synchronous Bus and Asynchronous Bus.

Synchronous Bus:-

- In synchronous bus, all devices derive timing information from a common clock line. Equally spaced pulses on this line define equal time intervals. In the simplest form of a synchronous bus, each of these intervals constitutes a bus cycle during which one data transfer can take place.

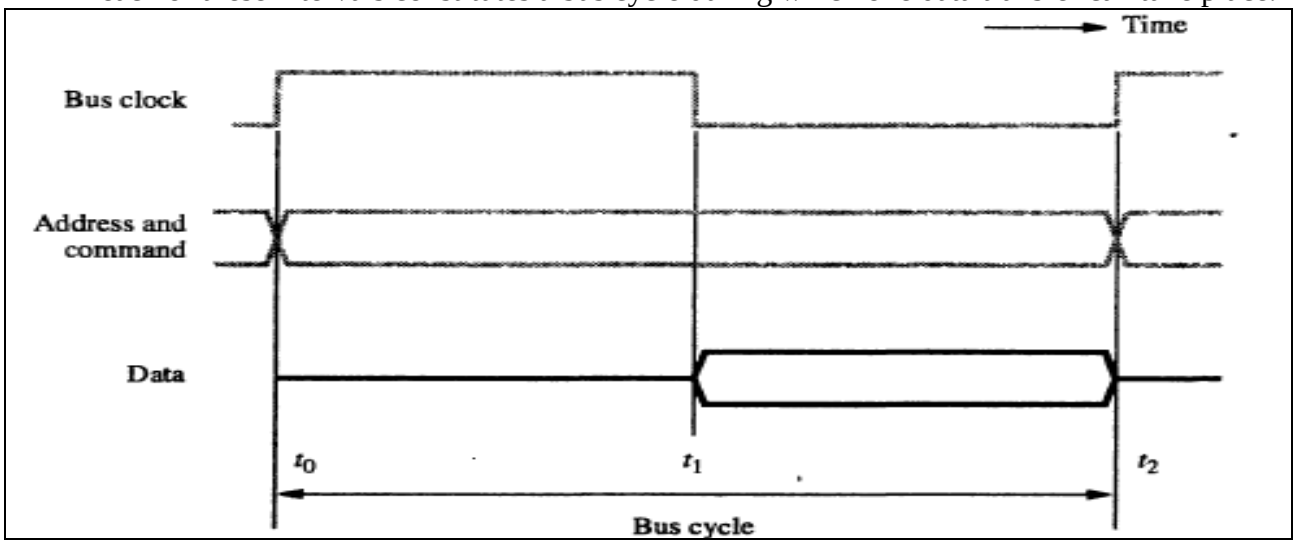


Figure 14: Timing of an input transfer on a synchronous bus

A sequence of events during a read operation:

- At time to, the master places the device address on the address lines and sends an appropriate command on the control lines. In this case, the command will indicate an input operation and specify the length of the operand to be read.
- The clock pulse width t1 – t0 must be longer than the maximum delay between devices connected to the bus. The clock pulse width should be long to allow the devices to decode the

address and control signals so that the addressed device can respond at time t1. The slaves take no action or place any data on the bus before t1.

- The picture shows two views of the signal except the clock. One view shows the signal seen by the master and the other is seen by the salve. The master sends the address and command signals on the rising edge at the beginning of clock period (t0).These signals do not actually appear on the bus until tam. Sometimes later, at tAS the signals reach the slave. The slave decodes the address and at t1, it sends the requested data.
- At t2, the master loads the data into its input buffer. Hence the period t2, tDM is the setup time for the masters input buffer. The data must be continued to be valid after t2 , for a period equal to the hold time of that buffers.

Demerits:

1. The device does not respond.
2. The error will not be detected.

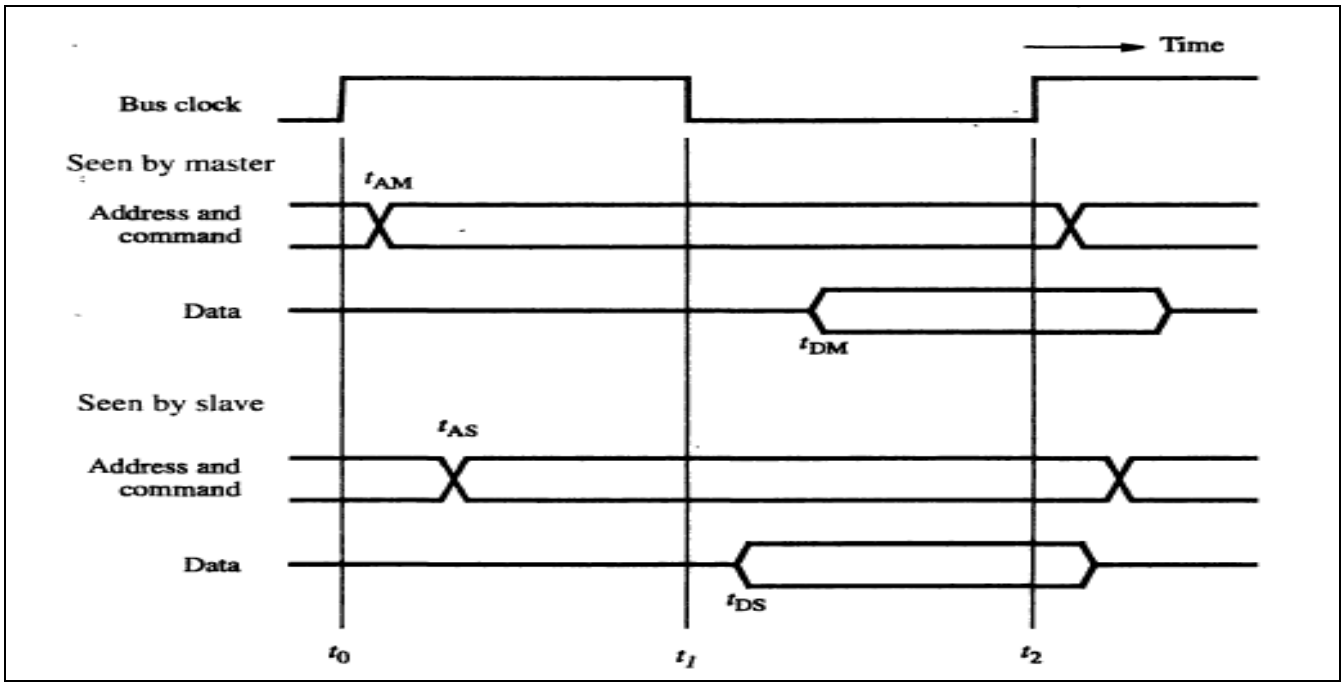


Figure 15: A detailed timing diagram for the input transfer of figure 14

Multiple Cycle Transfer:-

- During, clock cycle1, the master sends address and command information. On the bus requesting a "read" operation. The slave receives this information and decodes it. At the active edge of the clock (i.e.) the beginning of clock cycel2, it makes accession to respond immediately.
- The data become ready and are placed in the bus at clock cycle3. At the same times, the slave asserts a control signal called "slave-ready". The master device has been waiting for this signal, strobes, and the data to its input buffer at the end of clock cycle3.
- The bus transfer operation is now complete & the master sends a new address to start a new transfer in clock cycle4. The „slave-ready" signal is an acknowledgement form the slave to the master confirming that valid data has been sent.

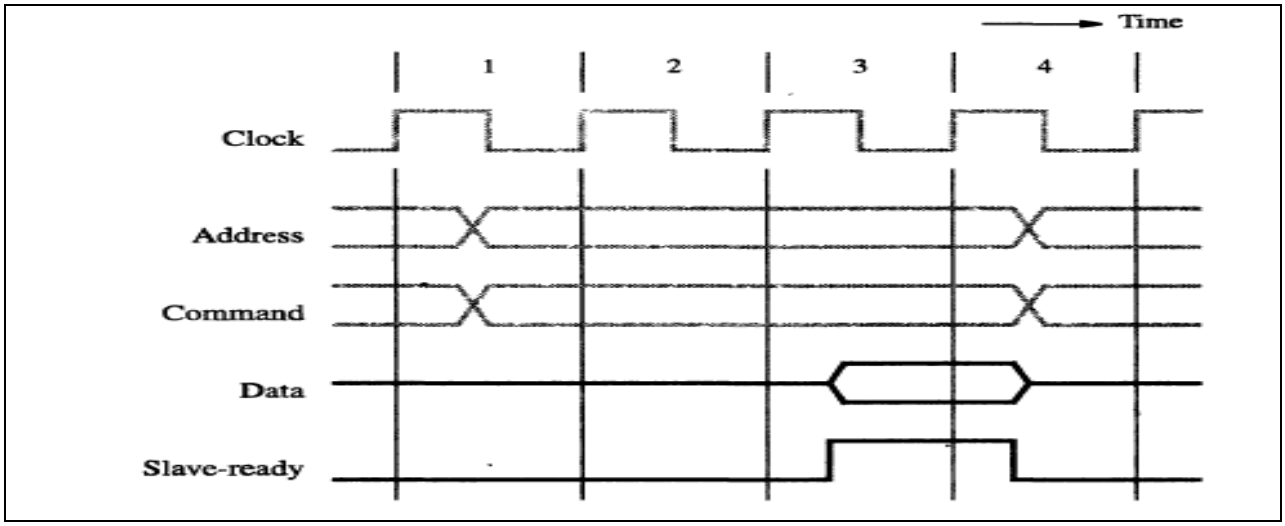


Figure 16: An input transfer using multiple clock cycles

Asynchronous Bus:-

An alternate scheme for controlling data transfer on. The bus is based on the use of "*handshake"* between *Master* and the *Slave*. The common clock is replaced by two timing control lines. They are Master–ready and Slave ready.

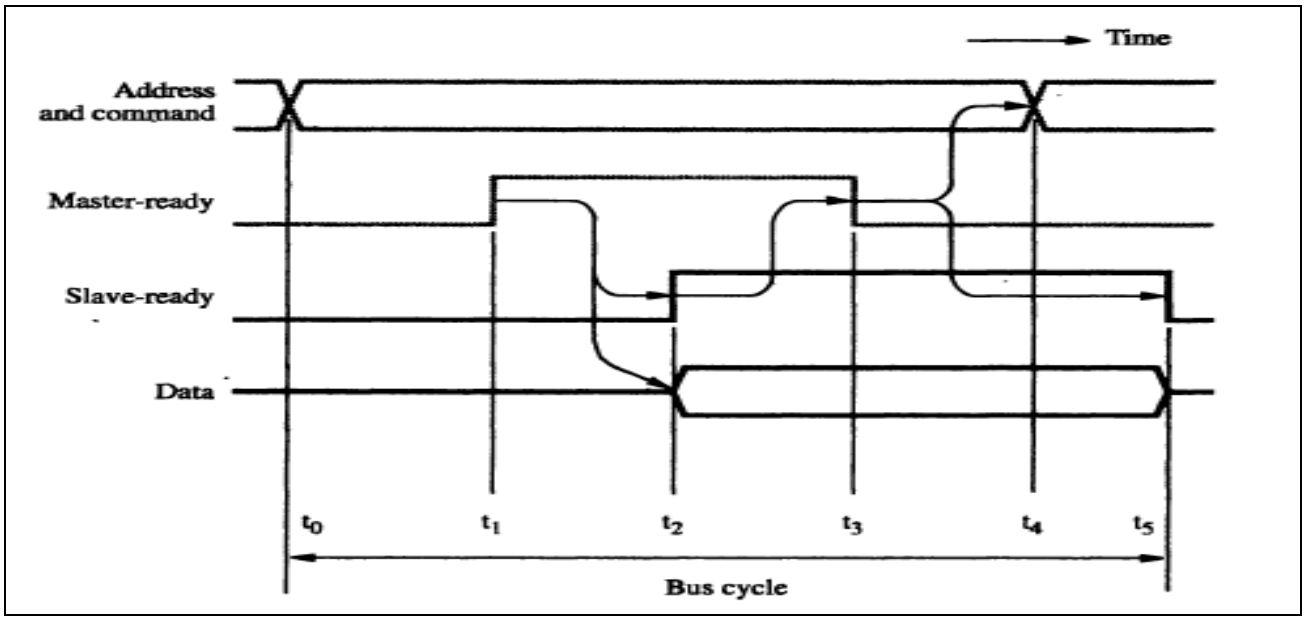


Figure 17: Handshake control of data transfer during an input operation

The handshake protocol proceeds as follows:

At t0*:* The master places the address and command information on the bus and all devices on the bus begin to decode the information.

At t1: The master sets the Master ready line to 1 to inform the I/O devices that the address and command information is ready.

- The delay t1 – t0 is intended to allow for any skew that may occur on the bus.

- The skew occurs when two signals simultaneously transmitted from one source arrive at the destination at different time.
- Thus to guarantee that the Master ready signal does not arrive at any device a head of the address and command information the delay t1 – t0 should be larger than the maximum possible bus skew.

At t2: The selected slave having decoded the address and command information performs the required i/p operation by placing the data from its data register on the data lines. At the same time, it sets the "slave – Ready" signal to 1.

At t3:The slave ready signal arrives at the master indicating that the input data are available on the bus.

At t4: The master removes the address and command information on the bus. The delay between t3 and t4 is again intended to allow for bus skew. Erroneous addressing may take place if the address, as seen by some device on the bus, starts to change while the master – ready signal is still equal to 1.

At t5: When the device interface receives the 1 to 0 transitions of the Master–ready signal, it removes the data and the slave – ready signal from the bus. This completes the input transfer.

In this diagram, the master place the output data on the data lines and at the same time it transmits the address and command information. The selected slave strobes the data to its output buffer when it receives the Master ready signal and it indicates this by setting the slave–ready signal to 1. At time t0 to t1 and from t3 to t4, the Master compensates for bus.

In the above and below timing diagrams, it is assumed that the master compensates for bus skew and address decoding delay. It introduces the delays from $\mathbf{t_0}$ to $\mathbf{t_1}$ and from $\mathbf{t_3}$ to $\mathbf{t_4}$ for this purpose. If this delay provides sufficient time for the I/O device interface to decode the address, the interface circuit can use the Master-ready signal directly to gate other signals to or from the bus,.

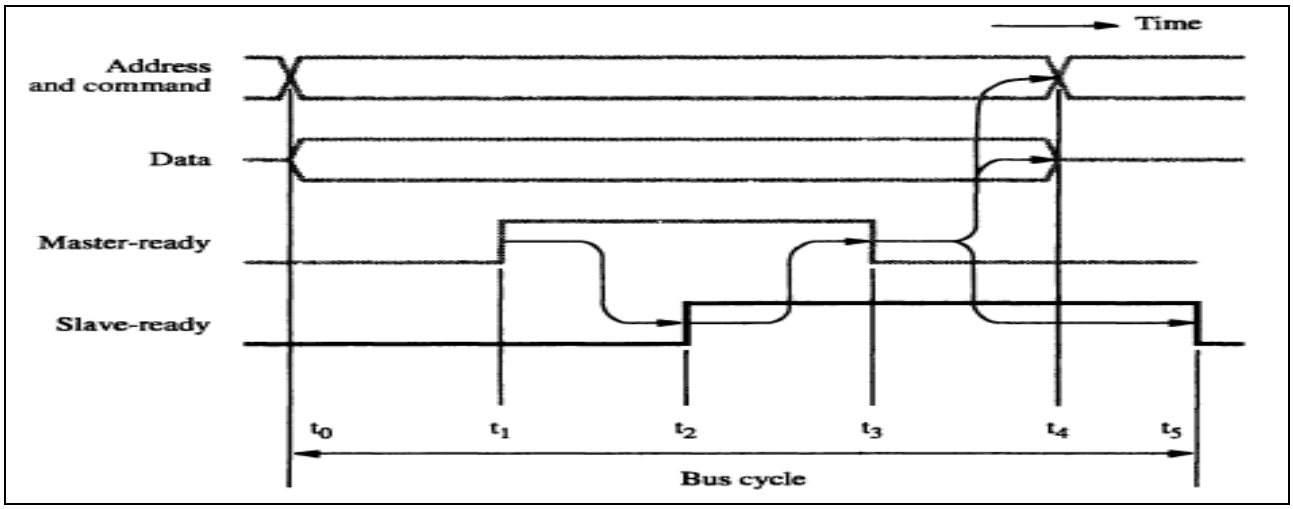


Figure: Handshake control of data transfer during an output operation

A change of state is one signal is followed by a change is the other signal. Hence this scheme is called as Full Handshake. It provides the higher degree of flexibility and reliability.

Advantages of asynchronous bus:

- Eliminates the need for synchronization between the sender and the receiver.
- Can accommodate varying delays automatically, using the Slave-ready signal.

Disadvantages of asynchronous bus:

- Data transfer rate with full handshake is limited by two-round trip delays.
- A data transfer using a synchronous bus involves only one round trip delay, and hence a synchronous bus can achieve faster rates.

Speed, Size and Cost:

- Big challenge in the design of a computer system is to provide a sufficiently large memory, with a reasonable speed at an affordable cost.

Static RAM:

- Very fast, but expensive, because a basic SRAM cell has a complex circuit with 6 transistors, making it impossible to pack a large number of cells onto a single chip.

It is impractical to build a large memory using SARAM chips because of cost.

Dynamic RAM

- Simpler basic cell circuit, hence are much less expensive, but significantly slower than SRAMs.

SRAM and DRAM are volatile.

Magnetic disks:

- Storage provided by DRAMs is higher than SRAMs, but is still less than what is necessary.
- Secondary storage such as magnetic disks provides a large amount of storage, but is much slower than DRAMs.

Conclusion:

- A huge amount of cost-effective storage can be provided by magnetic disks.
- Main memory is built using DRAM
- Cache memory is fast and smaller units built using SRAM.

All of these different types of memory units are employed effectively in a computer.

The entire computer memory can be viewed as the hierarchy depicted in below figure.

- The fastest access is to the data held in processor register. Registers are at the top of the memory hierarchy.
- Relatively small amount of memory that can be implemented on the processor chip. This is called processor cache.
- Two levels of cache. A Primary cache Level (L1) cache is on the processor chip. Secondary cache Level (L2) cache is in between main memory and processor. This is implemented using SRAM.

- Next level is main memory, implemented using DRAM , in the form of SIMMs. Much larger, but much slower than cache memory.
- Next level is magnetic disks. Huge amount of inexpensive storage.
- Speed of memory access is critical, the idea is to bring instructions and data that will be used in the near future as close to the processor as possible.
- Disk devices provide huge amount of inexpensive storage.

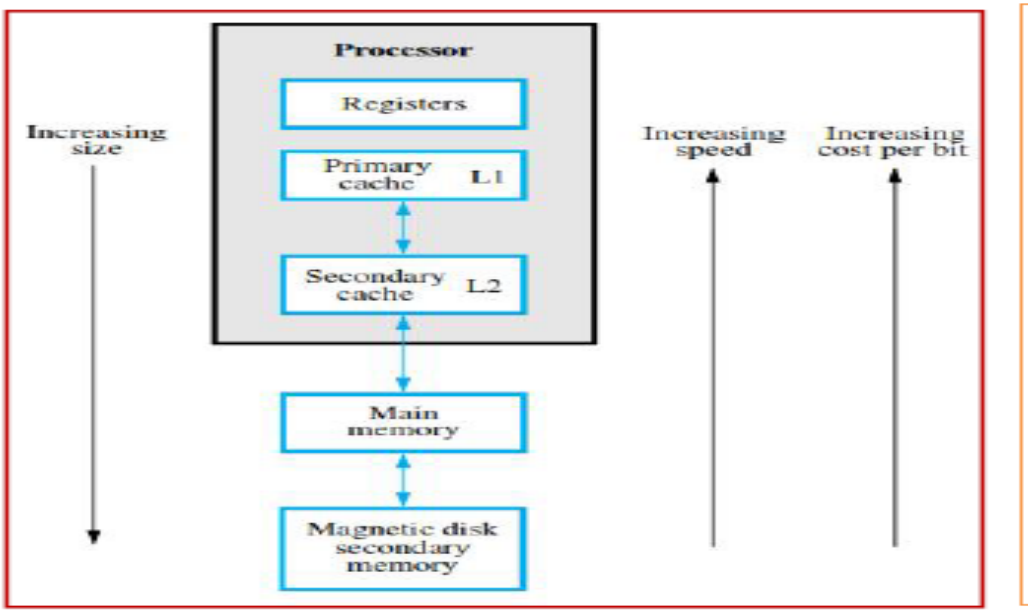


Figure: Memory Hierarchy

- The main memory is implemented using the dynamic components (**SIMM (Single In-line Memory Module), RIMM (Rambus In-line Memory Module), and DIMM (Dual In-line Memory Module)**).
- The access time for main memory is about 10 times longer than the access time for L1 cache.

5. Cache Memories

The cache is a small and very fast memory, interposed between the processor and the main memory. Its purpose is to make the main memory appear to the processor to be much faster than it actually is. The effectiveness of this approach is based on a property of computer programs called *locality of reference*.

Locality of Reference:

- ✓ Many instructions in the localized areas of the program are executed repeatedly during some time period and remainder of the program is accessed relatively infrequently.
- ✓ It manifests itself in 2 ways. They are *temporal* and *spatial*.
  - ★ *Temporal*
    - The recently executed instructions are likely to be executed again very soon.
  - ★ *Spatial*
    - The instructions in close proximity to recently executed instruction are also likely to be executed soon.
- ✓ If the active segment of the program is placed in cache memory, then the total execution time can be reduced significantly.
- ✓ The term Block refers to the set of contiguous address locations of some size.
- ✓ The cache line is used to refer to the cache block
- ✓ The Cache memory stores a reasonable number of blocks at a given time but this number is small compared to the total number of blocks available in Main Memory.
- ✓ The correspondence between main memory block and the block in cache memory is specified by a mapping function.
- ✓ The Cache control hardware decides that which block should be removed to create space for the new block that contains the referenced word.
- ✓ The collection of rule for making this decision is called the *replacement algorithm*.

- The cache control circuit determines whether the requested word currently exists in the cache. If it exists, then Read/Write operation will take place on appropriate cache location. In this case
- *Read/Write hit* will occur.
- In a Read operation, the memory will not involve.
- The write operation is proceeding in 2 ways. They are Write-through protocol and Write-back protocol

Write-through protocol:

- Here the cache location and the main memory locations are updated simultaneously.

- Data is placed in the cache and immediately written on to the disk . Acknowledgment is sent to the processor. Response time is longer and risk of data loss is low.

Write-back protocol or copy-back protocol:

- This technique is to update only the cache location and to mark it as with associated flag bit called *dirty/modified bit.*
- *Dirty bit indicates whether data in cache is committed to disk or not.*
- The word in the main memory will be updated later, when the block containing this marked word is to be removed from the cache to make room for a new block. This technique is known as write-back or copy-back protocol.
- Data is placed in cache and acknowledgement is sent to host immediately. Later data is written to disk. Write response time is faster.
- If the requested word currently not exists in the cache during read operation, then *read miss* will occur.
- To overcome the read miss *Load –through or early restart protocol* is used.

Read Miss:

- The block of words that contains the requested word is copied from the main memory into cache.

Load –through:

- After the entire block is loaded into cache, the particular word requested is forwarded to the processor. If the requested word not exists in the cache during write operation, then *Write Miss* will occur. If Write through protocol is used, the information is written directly into main memory.
- If Write back protocol is used then blocks containing the addressed word is first brought into the cache and then the desired word in the cache is over-written with the ne

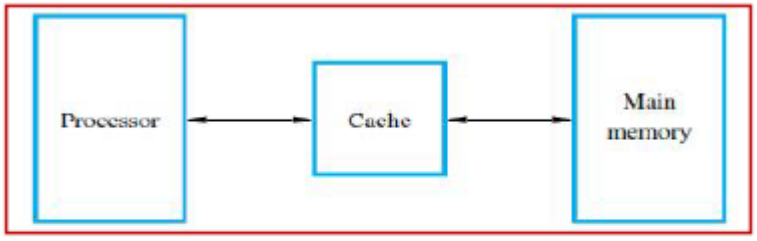


Figure 14: Use of cache memory

Cache Memories

Mapping Function:

Since main memory is large compared to the cache, care must be taken while mapping the blocks from main memory into cache. There are 3 techniques are there for mapping function.

Direct Mapping:

Consider a cache memory having 128 blocks of 16 words each. Total 2048 (2K) words. Main memory consists of 16-bit address contains 64K words are organized as 4K blocks*16 words.

- It is the simplest technique in which block j of the main memory maps onto block "j" modulo 128 of the cache.
- Thus whenever one of the main memory blocks 0,128,256 is loaded in the cache, it is stored in block 0.
- Block 1,129,257 is stored in cache block 1 and so on.
- 32 Main memory blocks are mapped onto one cache block.
- The contention may arise when the cache is full
- When more than one memory block is mapped onto a given cache block position

The contention is resolved by allowing the new blocks to overwrite the currently resident block. Placement of block in the cache is determined from memory address.

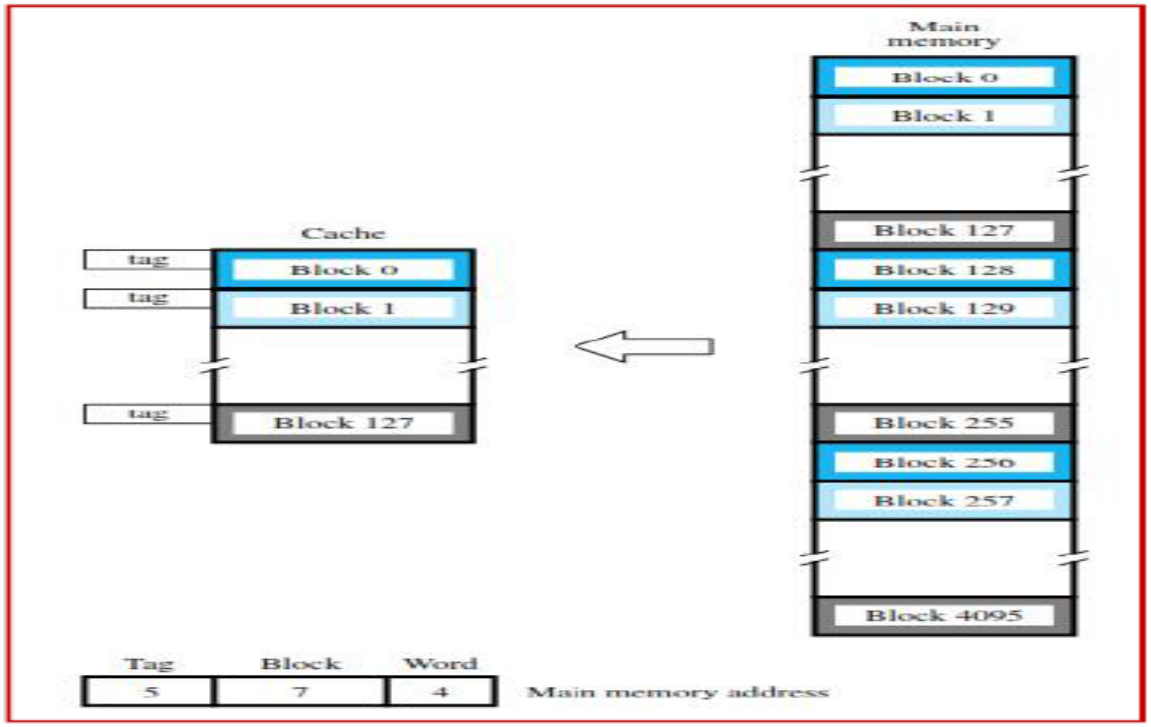


Figure 15: Direct-mapped cache

- The memory address is divided into 3 fields. They are,
- ***Low Order 4 bit field (word):*** selects one of 16 words in a block.
- **7 bit cache block address field:** When new block enters cache, 7 bit determines the cache position in which new block must be stored out of 128.
- **5 bit Tag field:** The high order 5 bits of the memory address of the block is stored in 5 tag bits associated with its location in the cache. They identify out of 32 blocks, which block currently resides in cache.
- As execution proceeds, the high order 5 bits of the address are compared with tag bits associated with that cache location.
- If they match, then the desired word is in that block of the cache.
- If there is no match, then the block containing the required word must be first read from the main memory and loaded into the cache.

Merit: It is easy to implement.

Demerits: It is not very flexible. Even though other blocks are free, identified particular blocks are overwritten.

Associative Mapping:

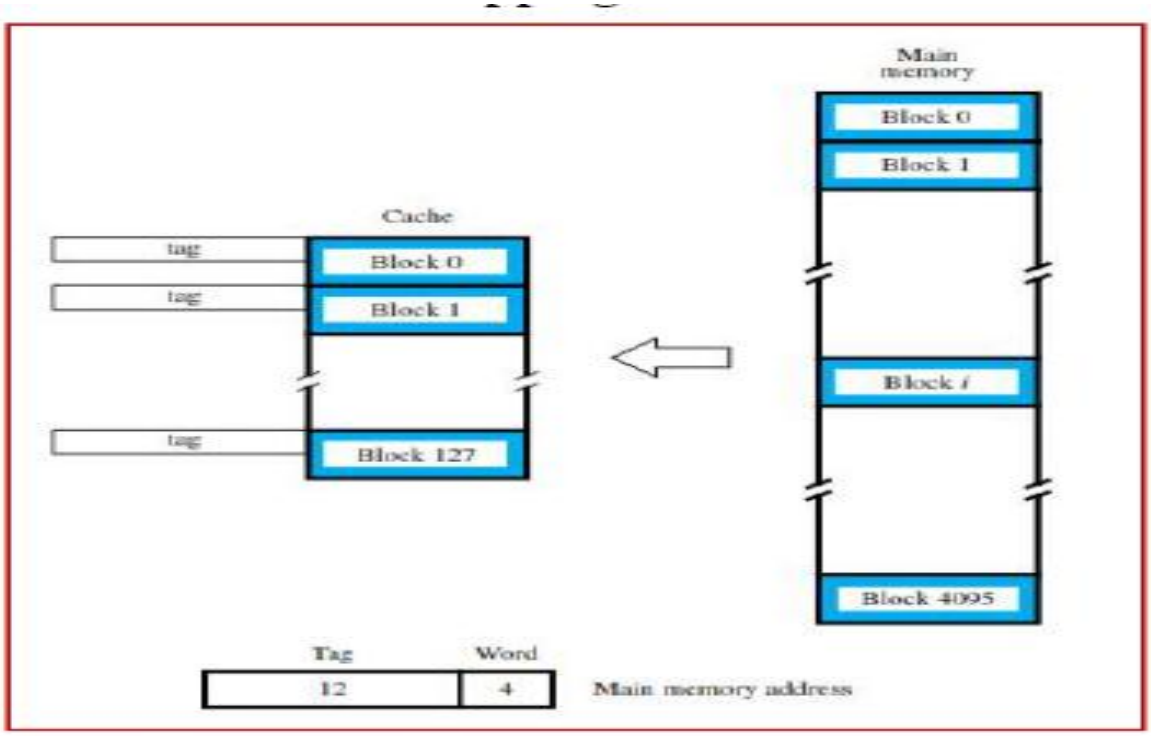


Figure 16: Associative-mapped cache

- In this method, the main memory block can be placed into any cache block position.
- No need to identify a particular block in cache when new block arrives. It can be placed in any free cache block.
- 12 tag bits will identify a memory block when it is resident in the cache. The tag bits of an address received from the processor are compared to the tag bits of each block of the cache to see if the desired block is present. This is called **associative mapping.**
- It gives complete freedom in choosing the cache location.
- A new block that has to be brought into the cache has to replace (eject) an existing block if the cache is full. In this method, the memory has to determine whether a given block is in the cache. A search of this kind is called an **associative Search.**

Merit: It is more flexible than direct mapping technique.

Demerit: Its cost is high.

Set-Associative Mapping:

- It is the combination of direct and associative mapping.
- The blocks of the cache are grouped into sets and the mapping allows a block of the main memory to reside in any block of the specified set.
- In this case, the cache has two blocks per set, so the memory blocks 0, 64,128……..4032 maps into cache set "0" and they can occupy either of the two block position within the set.

- Totally three are 64 sets, hence 6 bit set field. Determines which set of cache contains the desired block.

- 6 bit tag field: The tag field of the address is compared to the tags of the two blocks of the set to clock if the desired block is present.

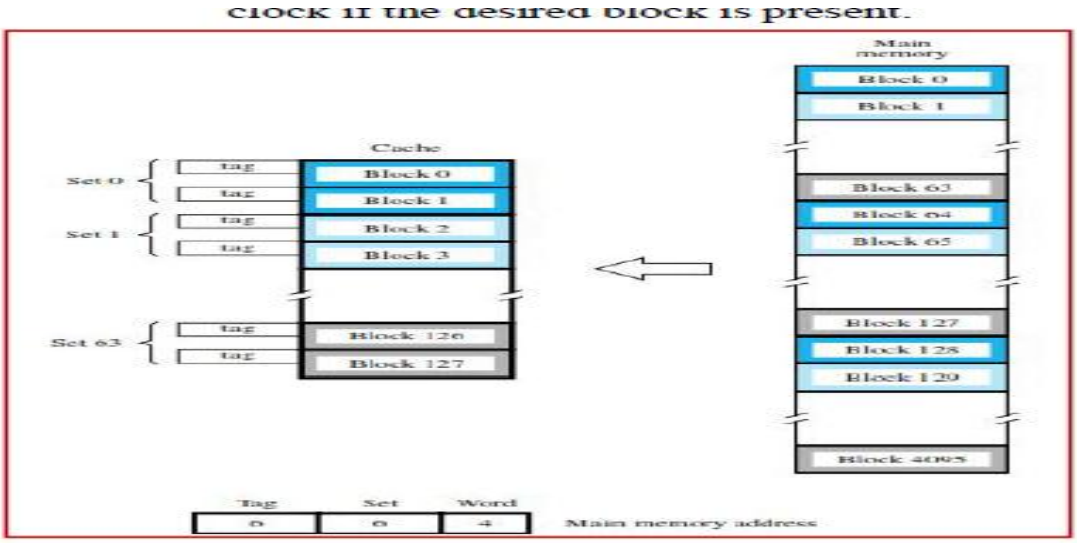


Figure 17: Set-associative-mapped cache with two blocks per set

- The cache which contains 1 block per set is called **direct Mapping.**
- A cache that has "k" blocks per set is called as "**k-way set associative cache"**.
- Each block contains a control bit called a **valid bit**.
- The Valid bit indicates that whether the block contains valid data.
- The dirty bit indicates that whether the block has been modified during its cache residency.
- Valid bit=0, When power is initially applied to system
- Valid bit =1, When the block is loaded from main memory at first time.
- If the main memory block is updated by a source & if the block in the source is already exists in the cache, then the valid bit will be cleared to "0". If Processor & DMA uses the same copies of data then it is called as the **Cache Coherence**

Merit:

- The Contention problem of direct mapping is solved by having few choices for block placement.
- The hardware cost is decreased by reducing the size of associative search.

Replacement Algorithm

- In direct mapping, the position of each block is pre-determined and there is no need of replacement strategy.
- In associative & set associative method, the block position is not pre determined; i.e. when the cache is full and if new blocks are brought into the cache, then the cache controller must decide which of the old blocks has to be replaced.
- Therefore, when a block is to be over-written, it is sensible to over-write the one that has gone the longest time without being referenced. This block is called **Least recently Used (LRU) block & the technique is called LRU algorithm.**

- The cache controllers track the references to all blocks with the help of block counter.

Example:

- Consider 4 blocks/set in set associative cache, 2 bit counter can be used for each block.
- When a **'hit'** occurs, then block counter=0; the counter with values originally lower than the referenced one are incremented by 1 and all others remain unchanged.
- When a **'miss'** occurs & if the set is full, the blocks with the counter value 3 is removed, the new block is put in its place & its counter is set to „0" and other block counters are incremented by 1.
- Merit: The performance of LRU algorithm is improved by randomness in deciding which block is to be over-written.