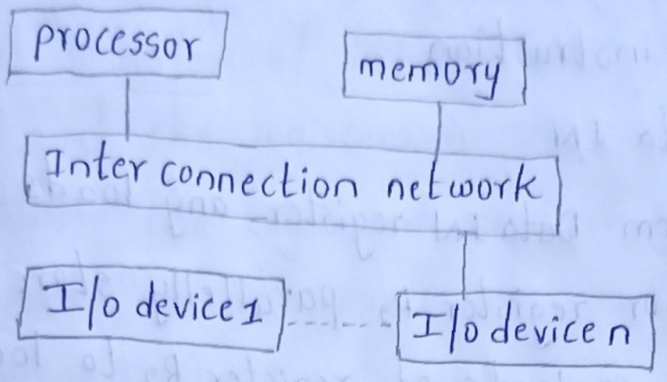
Unit-3

5th chapter

Basic Input/output

Accessing I/O Devices:-



* The components of a computer system communicate with each other through an interconnection network
* The interconnection network consists of circuits needed to transfer information between the processor, the memory unit, and a no. of I/O devices
* Each I/O device is not taken as a seprate device rather the I/O devices are as an addressable locations to the processor considered
* Each I/O device is mapped with one memory address interface

memory-mapped I/O:- The I/O devices is a way to exchange data and instructions between cpu and I/O devices attached to it

* memory-mapped I/O is one where the memory and the I/O devices share the same address space

With memory-mapped I/O, any instruction that can access memory can be used to transfer data to (or) from an I/o device

ex:- if data IN is the address of a register in an input device the instruction

Load R_2, Data IN

reads the data from Data IN register any loads them in to processor register R_2. parallelly store R_2, Data out send the contents of register R_2 to location DATA OUT which is a register in an output device

I/o device Interface:-

I/o devices are connected to a inter connection network by using circuit called the device interface

* which provides the means data transfer between cpu and I/o devices.

(ie Computer system includes special Hardware components between the cpu and I/o to supervise and synchronize all I/o transfer. These components called interface)

* The interface includes some registers that can be accessed by the processor

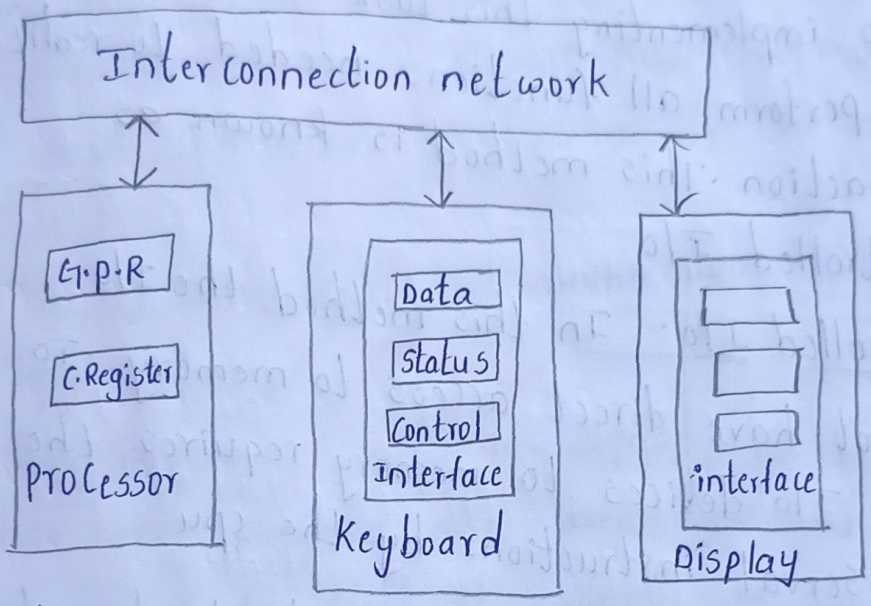
data register:- may serve as a buffer for data transfer

Status register:- may hold info about the current status of the device

Control register:- may store the info that controls the behaviour of the device operational

* These registers are accessed by program instructions
* Typical transfer of information between I/o registers and the registers in the processor

fig shows how the keyboard and display devices are connected to the processor from the software point of view



* 2 techniques are there to allocate addresses to memory and I/o devices.
* Some addresses are assigned to memories and some to I/o devices.
* An I/o device is also treates as to memory location and one address is assigned to it

In memory mapped I/o only one address space that is shared by both memory and I/o (No address overlapping)

* Address space is defined as all possible address that processor can genrate. some addresses are assigned to memory and some to I/o devices.

Program - controlled I/o :- (modes of data transfer)

* Consider a task that reads characters typed on a keyboard, stores these data in memory, and displays the same characters on a display screen

* In order to implementing this task. write a program that perform all functions needed to realize the desired action. This method is known as program - controlled I/o.

Program - controlled I/o :- In this method the I/o devices does not have direct access to memory. To Transfer from I/o devices to memory requires the execution of serval instructions by the cpu

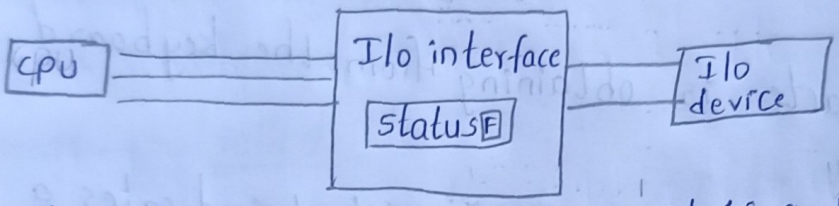
* To Transfer each character from the keyboard into the memory and then to the display it is necessary to ensure that this happens at the right time

* An input character must be read in response to a key being pressed.

* for monitor the character must be sent to only when the display device is able to accept it
* The I/o devices are electro mechanical or electro magnetic devices but cpu is electronic devices
* so the data transfer rate of I/o devices is slower than the transfer rate of the cpu so synchronization mechanism is needed
i.e A mechanism is required to synchronize the transfer of data between them
one solution to this problem is signalling protocol
ex:- on output, the processor sends the first character and then waits for a signal from the monitor to Send next character. it then sends the 2nd character and so on.
* An input character is obtaining from the keyboard in a similar way.
i.e. The processor waits for signal that indicates a key has been pressed and that a binary code corresponding to that character is available in an I/o register associated with the keyboard. Then the cpu proceeds to read the code
* The keyboard has interface it responds to a key being pressed and convert that character is binary code that can be used by the Computer
* we assume that the ASCII code is used. The In which each character code occupies one byte (8 bits)

* Let KBD-DATA be the address label of an 8 bit register that holds the genrated character
* 8 bit status register whose address label is KBD-STATUS. In this register one bit called KIN when user pressed keyboard KIN ← 1
* The processor can read the status flag KIN to determine when a character code has been placed in KBD-DATA
* when the processor read the status register flag to determine its state, call it as processor polls the I/o device

Polling:-

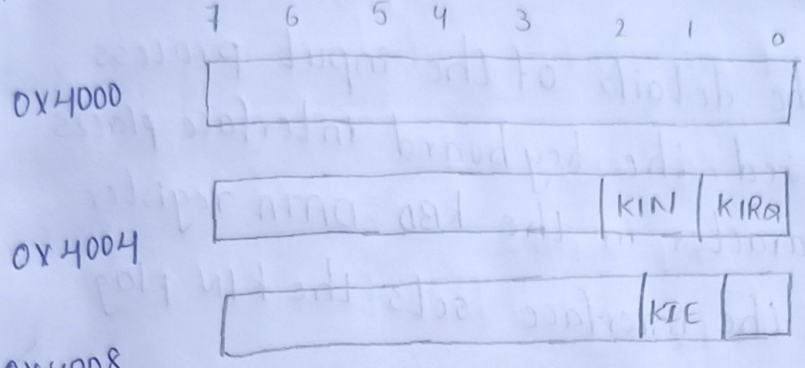


* polling is not allow mechanism it is a protocol in which cpu continously checks the Flag bit of status registers whether the devices need attention
(Here cpu continious check whether the device needs attention)
i.e The cpu will check the flag bit continously for each transfer cpu is wasting device time while checking the flag instead of doing some other useful work

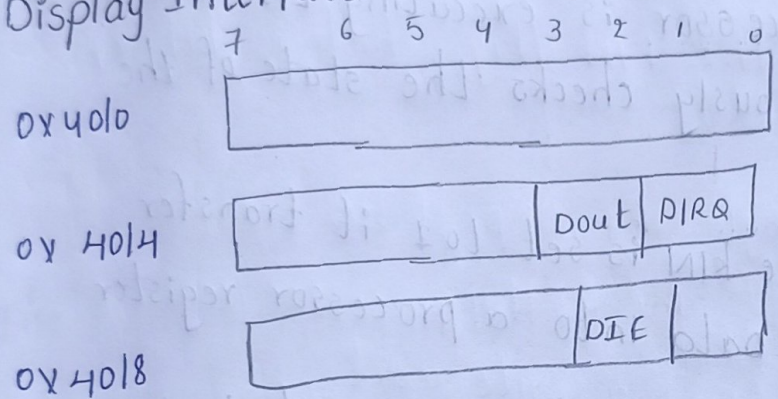
* The display includes on 8 bit register which we call DISP-DATA used to receive character from the processor
* it has status flag in status register called Dout which is 1 bit in a status register DISP-STATUS

Registers in keyboard and display interfaces:-

Addresses keyboard Interface



Display Interface



The interface for each device also includes a control registers

* The registers in I/o devi interfaces are to be accessed as if they are memory locations so Assign specific address to each register. each register address is specified in Hexa decimal format (All addresses are word-aligned)

* A program is needed to perform the task of reading the characters produced by the keyboard storing them in memory and sending them to the display
* To perform I/o transfers the processor must execute machine instructions that check the state of the status flags and transfer data bit the processor and I/o devices.
* Let us consider the details of the input process when a key is pressed the keyboard interface places the ASCII encode character in the KBD-DATA register
* At the some time the interface sets the KIN Flag to 1
* mean while the processor is executing the I/o program which continuously checks the state of the KIN flag.
* when it detects the KIN is set to 1 it transfer the contents of KBD-Data into a processor register
* once the content of KBD-Data are read, KIN must be closed to 0, which is done automatically by interface circuit
* if a second character is entered at the keyboard KIN is again set to 1 and the process repeats

The desired action can be carry out by following operations

Read wait

Read the KIN flag

Branch to read wait if kIN=0
Transfer data from kBD-Data to R5
which reads the character in to processor register R5
* process takes place when characters are transferred from the processor to the display
when DOUT is equal to 1 the display is ready to receive a character
* The processor monitors DOUT and when DOUT is equal to 1 the processor transfers the ASCII character to DISP-DATA Then clears Dout to "0" automatically when the display device is ready to receive a character
Dout is again set to 1
This can be achieved by performing the operations

WRITE WAIT
Read the Dout flag
Branch to write wait if Dout=0
Transfer data from R5 to DISP_Data

The wait loop is executed repeated until the status flag Dout is set to 1
* we assume that the intial state of kIN is 0 and intial state of Dout is 1
* Computer uses memory-mapped I/o in which some address are used to refer to registers in I/o interfaces, data can be transfered between these registers and processor using Load and store and move

Load Byte R5, KBD-Data

Contents the keyboard character buffer KBD-Data can be transfered to register R5 in the processor by the above instruction

store Byte R5, DISP-Data

Here load Byte & store byte instructions used for Byte operand ie operand size is byte

Load and store used for word operands

Read wait Load Byte R4, KBD-status

And R4, R4 #2

Branch-if [R4]=0 Read wait

Load Byte R5, KBD-Data

And is used to test the KIN flag, which is bit b_1 of the status

write wait Load Byte R4, DISP-status

And R4, R4 #4

Branch-if [R4]=0 write wait

Store Byte R5, DISP-Data

Here the data transfer rate of cpu is very fast but I/o devices is very slow since this type of transfer is in efficient

what is an interrupt

Distrubing the cpu what is presenting doing and transfer the control some where else

i.e when interrupt can occurs the control transfers from currently running program to another service routine Again the control returns to the original program after the service program is executed

Difference between interrupt and subroutine call

i) Interrupts are initiated by signals rather than from the execution of an instruction

ii) The address of ISR is determined by the Hardware rather than from the address field of an instruction

cpu also have one status register called processor status Register

what happen when interrupt occours

it is same way as subroutine call

whenever interrupts occours the pc value and present status of cpu is stored in stack and cpu

* whenever an interrupt occours the cpu completes the execution of the current instruction and save pc and then starts the execution ISR.

Interrupt - initiated I/O :-

In the programmed I/o the program constantly monitors the device status whether it is ready for data transfer

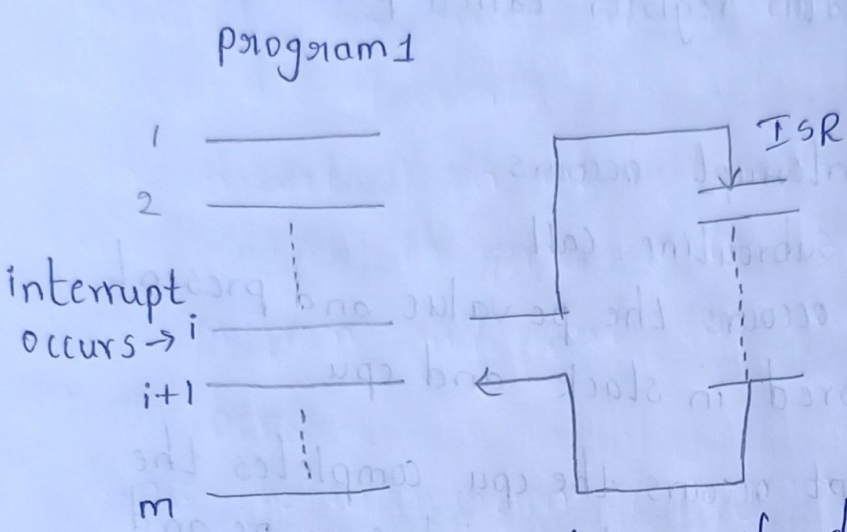
it keeps the cpu busy need lessly.

it can be avoided by making the interface continuously monitor the device status and raise an interrupt to the cpu as soon as the device is

ready for data transfer

* upon detecting external interrupt signal, processor stops what is presently doing and branches to an interrupt service routine (ISR) to initiate Iprocess the I/o transfer and return to the task it was originally doing

* In interrupt-initiated I/o cpu performs data transfer but it not involving in checking whether the device is ready for data transfer or not



* In Interrupt-initiated I/o the device controller should have some additional intelligence for checking the status of devices and raising an interrupt whenever data transfer is required

* A treatment of an ISR is very similar to that of a subroutine - Before starting execution of the ISR status information and contents of processor registers must be saved. this saved information must be restored before execution of the interrupted program is resumed In this way the orginal program can

continue execution without being affectly in any way by the interruption, except for the time delay.

* The task of saving any restoring information can be done automatically by the processor

* most modern processors save only the min-amount of information because the process the saving and restoring registers involves memory transfer the increase total execution time

Interrupt Latency:- The delay between the time can interrupt request is received and the start of the execution of ISR

* In some applications a long interrupt latency is un acceptable for these reasons the amount of information saved automatically by the processor when an interrupt request is accepted should be kept to a minimum (i.e processor saves only the content of pc and the processor_status register)

* In some earlier processors small no of registers are there, all registers are saved automatically by the processor Hard ware when an interrupt request is accepted

* Some computers provide 2 types of interrupts

* one saves all register contents and the other does not

* A particular I/o device may use either type depending up on its response time requirements
* Another approach is to provide duplicate sets of processor registers In this case a different set of registers can be used by the ISR these eliminating the need to save and restore registers
* The duplicate registers are called the shadow registers

Enabling and Disabling Interrupts:-

* Interrupts can arrive at any time they can after the sequence of events
* The arrival of an interrupt request from an external devices causes the processor to suspend the execution of one program and start the execution must be of an another
* Hence the interruption of program execution must be controlled carefully
  i-e we need a mechanism to enable and disable such interrupts as desired
* it is convenient to enable and disable interrupts at both the processor and I/o device ends
* The processor can either accept (or) ignore interrupt requests
* An I/o device can be allowed to raise interrupt requests (or) prevented from doing so

* A commonly used mechanism to achieve this is to use some control bits in registers
* The processor has a status register (PS), which contains information about its current state of operation
Let one bit, IE of this register be assigned for enabling/ disabling interrupts.
* Then the programmer can set (or) clear IE to cause the desired action
when IE=1 the processor responds to the interrupt
when IE=0 the cpu ignores the interrupt request from I/O devices.
* The program must issue enable interrupt (EI) instruction to set the IE flag
The cpu sets the IE flag when executing this instruction (1)
* The program must issue disable interrupt (DI) instruction to reset the IE flag
The cpu reset (0) the IE flag when executing this instruction
i.e the cpu's behaviour of serviceing interrupt is controlled by the program that is being executed currently
These is one special situation when the cpu resets IE flag on its own
i.e During interrupt servicing the cpu resets the IE flag immediately after saving the return address

in to memory or in to register and before branching to the ISR Thus when cpu starts execution of ISR interrupts are disabled
There fore it is upto ISR to allow interrupts (by EI) if it wishes

* The interface of an I/o device includes a control register one bit in this register may be dedicated to to interrupt control
The I/o devices is allowed to raise interrupt requests only when this bit is set to 1 (KIE) is set to 1 (keyboard interrupt enable)
Let us summarize the sequence of events involved in Handling an interrupts request from a single device

1) The device raises an interrupt request
2) The processor interrupts the program currently being executed and saves the contents of the PC and ps registers
3) Interrupts are disabled by clearing the IE bit in the ps to 0
4) The action required by interrupt is performed by the interrupt-service routine, during the cpu inform to device that its request has been recognizes and in response it deactives the interrupt-request signal [when device activates the interrupt-request

Signal, it keeps this signal activated until it learns that the processor has accepted its request [i.e INTR signal will be active during execution of ISR]

5) upon completion of the ISR, the saved contents of the PC and PS registers are restored (enabling interrupts by set IC bit to 1) any execution of the interrupted program is restored

Handling multiple devices:-

when more than one devices raises an interrupt request signal the additional info is needed

for example device x may request an interrupt while an interrupt caused by device y is being serviced

(or) several devices may requests interrupts at exactly the same time Here rise a no. of questions

1) How can the processor determine which device is requesting an interrupt

2) How can the processor obtain the starting address of the appropriate routine in each case?

3) should a device be allowed to interrupt the processor while another interrupt is being serviced?

4) How should 2 (or) more simultaneous interrupt requests be handled.?

The following methods are used to decide which device to select

1) polling, vector interrupts and interrupt handling nesting.

Polling (Interrupt identification by polling)

when device request an interrupt to determine which device request an interrupt the information is availabl in its status registers.

* when the device raises an interrupt request it's self its IRQ bit ← 1 in its status registers whenever interrupt occours to determine which device raise it these is a special ISR that poll all I/o devices in the System.

* The first device encountered with its IRQ bit set to 1 is the device that should be serviced

An appropriate subroutine is then called to provide the requested service

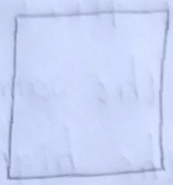
The polling scheme is easy to implement

Dis advantages:- is a lot of time is wasted by interrogating the IRQ bit of all devices.

2) vector interrupts:-

In vector interrupts a device requesting an interrupt identifies itself directly by sending a special code to the processor over the interconnection network i-e This enables the processor to identify the device that genrated the interrupt Then processor determine the memory address of the required interrupt - SR these allocate an area in the memory permanently

to hold the address of ISR. These address are usually referred to as Interrupt vector table
when interrupt request arrives the info provided by requesting device is used as a pointer to IVT and the address in wrapping IV is loaded in to pc



32 interrupts
128 bytes allocated to it

3) interrupt Nesting:-

During the execution of one ISR if it allows another interrupt then this known as interrupt nesting

ex:- suppose the cpu is intially execuling program 'A' when the first interrupt occours.
Then cpu stores return address of A and starts executing ISRI Now say in the mean time second internet occours The cpu again after storing return address of instruction in ISRI. start executing ISR_2 ..... after completing ISR_2 the cpu resumes the execution of ISR_1 Similarly after ISR_1 completion resumes the execution of A

* The I/o devices should be organized in a priority structure. An interrupt request from a high priority device should be accepted while the processor is Servicing a request from a lower-priority device
Here assign a priority level the devices!
during the execution of ISR interrupt request will

be accepled from some devices but not from others depending up on the devices priority

Simultaneous Requests:-

There is also possibility that serval sources may request interrupt service simultaneously. In this case the processor must decide when to service first i.e when 2 devices interrupt the cpu at the same time the cpu services the device with the higher priority first

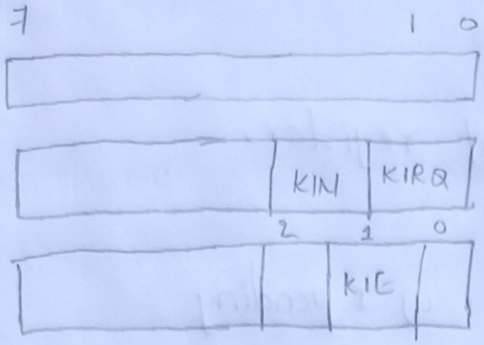
Controlling I/o device behaviour:-

All the devices connected to system is capable of genrate interrupts, we need a mechanism to control whether device is allowed to interrupt the processor

* A commonly used approach is to provide a control register in the device interface which holds the information needed to control the behavior of the device. The register is accessed as an addressable location one bit in this registers serves as the interrupt enable bit (IE)

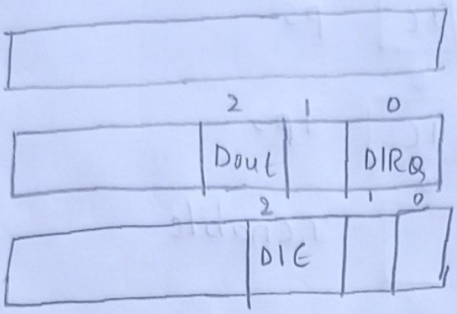
* when it is set to 1 an instruction that writes how info in to C.R, the device is allowed to interrupt the processor whenever it is ready for I/o transfer

ex:- fig shows the registers that may be used in the interface of keyboard and display devices.



KBD
Disp - Data
KBD - status
KBD - cont

Display interface



Disp - Data
Disp - status
Disp - cont

* Counter registers are used by cpu to control the device Bits in this register may be write only. only 1 (or) 2 bits are needed in handling I/o transfer the remaining bits can be used to specify other aspets of operation of a device (or) ignored if they are not needed

status - register has 2 bits KIN & KIRQ

KIRQ ← 1 if an interrupt request has been raised but not yet serviced

The keyboard may raise interrupt only when the IE bit i.e, KIE in its control register is to set to 1 i.e when both KIE & KIN are 1 then only KIRQ is set to 1

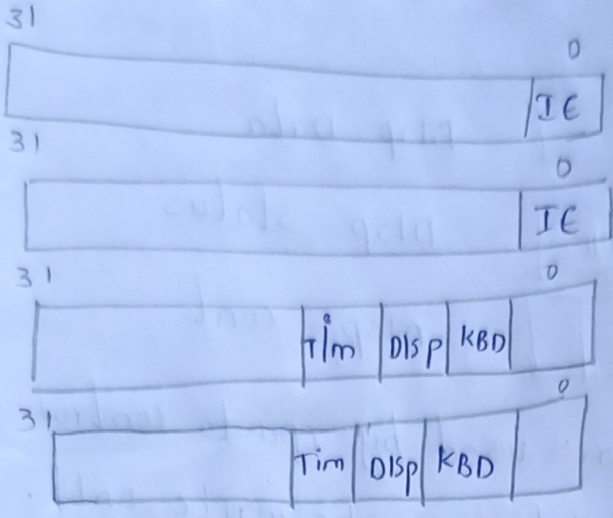
parallelly the DIRQ bit in status register of display interface indicates whether an interrupt request has been raised DIE in control register of this interface is used to enable interrupts.

The processor control registers:-

They are 4 processor control registers

① The status registers ps

② The Ips ③ I Enable 4) I pending



Ps registers include interrupt-enable bit (IE) in addition to other status information

The processor will accept interrupts only when this bit is set to 1

The Ips register is used to automatically save the contents of ps when an interrupt request is received and accepted

At the end of ISR the previous state of the processor is automatically restored by transffering the contents of Ips in to ps if nested interrupts are allowed then save contents Ips on the stack.

The I Enable register allows the processor to selectively respond to individual I/o device

A bit may be assigned for each device as in fig. when a bit is set to 1 the processor will accept interrupt request from the corresponding device

The Ipending register indicates the active interrupt requests. This is useful when multiple devices may raise requests at the same time then a program can decide which interrupt should be serviced first

These all registers are 32 bit registers

The registers cannot ~~a~~be accessed in same way as the genral purpose registers they cannot be accesed by Arithmetic and logic instructions they also cannot be accessed by Load & writ store instructions

Special addressing modes may be provided to acess the processor control registers. In RISC - style processor the special instructions may be of type

move Control R_2, PS

move Control IEnable, R_3