

INTERDATA (MODEL 74) INTERRUPT STRUCTURE

An interrupt is a break in the normal sequence of execution of instructions arising out of an event which is not directly under program control. Subject to certain global control parameters, an interrupt may occur at the conclusion of execution of any instruction. It is important to appreciate that the events concerned generally give rise, in the first instance, to an interrupt request to the processor. An interrupt request is said to be granted by the processor when the corresponding interrupt occurs, this may be some time after the request arose.

When an interrupt occurs it is necessary to be able to store all (or sufficient) of the current processor context (i.e. all program addressable registers) for it to be possible for the interrupted activity to be resumed at some later time. Since interrupts may occur frequently in a running system it is necessary for programs to be able to discover quickly the cause of an interrupt. Generally a complete context switch will be involved, that is all the program addressable registers of the interrupted activity must be preserved and a complete new context loaded. The source of the interrupt request will be the principal factor in deciding which new context is to be loaded.

The remainder of these notes is devoted to describing the various sources of interrupt request which exist in an Interdata 74 and the varying degrees to which the hardware assists in making the appropriate context switch.

Interrupt Mechanism

The basic action which takes place when an interrupt request is granted is to exchange the program status word (32 bits). A 4 halfword area in central memory is involved into which the current PSW is placed and from which the new PSW is taken. The layout of a 4 halfword area is as follows:-

address	contents
n	: old (i.e. immediately prior to interrupt) PSW status
n+2	: old location counter
n+4	: new (i.e. to be loaded at interrupt) PSW status
n+6	: new location counter

Which particular exchange area in memory is used is determined solely by the source of the interrupt request. There are several exchange areas built into the hardware of the machine, they all occur in low memory addresses. There is a direct correspondence between the classes (types) of interrupt request distinguished by the processor and the number of PSW exchange areas which exist.

Note that the hardware does not attempt to make a complete context switch, including, that is, all the general purpose registers. From now on in this document the term interrupt will be used in the Interdata sense of a PSW exchange.

It is necessary for programs to have some measure of control over the circumstances in which the processor will grant an interrupt request. In particular, it is necessary to be able to prevent the granting of certain requests altogether, while some more urgent activity is completed. In order to understand how this is done, and also to give some guide to considering each possible interrupt class in detail it is convenient to consider:-

The Role Of the PSW (Status Part)

Bits in the status part of the PSW determine the processor's response to an interrupt request. Somewhat simplified, there can be considered to be one bit in the status (PSW bits 15-8) corresponding to each interrupt class. If a bit is set then requests in that class are granted, this situation is described by saying the interrupt is enabled. If a bit is reset then interrupt requests of its class are disabled, that is they are not granted, although they may remain pending.

The mechanism by which an interrupt request is cleared (i.e. removed) differs according to its class, but generally the request may still exist after the PSW exchange. It is therefore usual that the new PSW of a given class has further interrupts of that class disabled, that is, the bit is reset.

The bits of PSW under consideration are indicated below, subsequently a detailed description is given of each in turn, in a fairly arbitrary order.

PSW (status) bit	meaning (when set)	
15	wait	8000
14	enable external interrupts	4000
13	enable machine malfunction interrupts	2000
12	enable fixed point divide fault interrupts	1000
11	enable automatic I/O service	800
10 & 9	not used in Model 74	
8	enable protected (user) mode	
7-4	not used	
3-0	condition code	

The Wait State (bit 15 set)

This is not an interrupt controlling bit at all. If it becomes set in the status word then the processor ceases executing instructions. In this state interrupts may still occur, i.e. it is not a halted state. In fact, I/O interrupts will occur even if the status bits are such as to disable them!!

Fixed Point Divide Fault (bit 12)

This interrupt may occur during a fixed point division operation if either an attempt is made to divide by zero or the quotient overflows. The interrupt request is generated only if bit 12 of the PSW is set. The request is always granted and cleared immediately. The relevant PSW exchange area is at location 48(hex).

Machine Malfunction - Close Down & Start Up (bit 13)

In order to appreciate the nature of this interrupt it is first necessary to know what happens when the processor undergoes close down and start up. A close down occurs as the final outcome of depressing the initialise (INT) switch or turning off the power at the control console switch. At this point the hardware preserves the processor context in (non-volatile) main memory. The 16 general purpose registers are stored in 16 successive locations in memory, the address of the first being contained in location 22(hex). The PSW is stored in locations 24 and 26(hex). (Locations 22-27 are known as the power fail locations.) When the processor comes out of the initialised state or when the power is restored, the reverse sequence, a start up, takes place. The registers are loaded from the 16 locations pointed at by the contents of location 22(hex) and the PSW is loaded from 24-27(hex).

After loading this context the processor is forced into the halted state (i.e. requiring manual intervention to start executing instructions) unless the processor is equipped with the auto-restart option. In the latter case the processor either starts executing instructions as determined by the PSW or halts, according to whether the console switches indicate the processor should be running or not. Some of the Interdata 74 processors are equipped with this option.

The effect of the machine malfunction (mmf) interrupt on this sequence of events is as follows. If it is enabled (bit 13 set) then an mmf interrupt occurs approximately 1 millisecond before a close down. Thus, the program has 1 ms in which to shut down activities in good order. On start up, again only if bit 13 enables it, an mmf interrupt is taken immediately after loading the preserved context. This gives the program a chance to re-start things in similar good order. The program can distinguish between the close down and start up mmf interrupt by the fact that the condition code is forced to zero on start up whereas the L bit is set on close down. The mmf interrupt exchange PSW area is located at 38(hex).

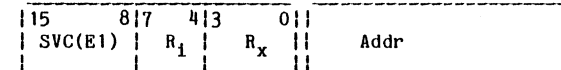
User Mode and the Illegal Instruction Interrupt (bit 8)

Many of the 256 possible opcode values do not have any meaning. If the processor attempts to execute one of these then an illegal instruction interrupt occurs. This interrupt may not be disabled, its PSW exchange area is located at 30(hex).

If bit 8 of the PSW is set then, effectively, all input or output and system control (i.e. those which explicitly address the PSW status) instructions become "illegal" as well. Thus, when bit 8 is set, an illegal instruction interrupt occurs if a program attempts to execute an I/O or system control instruction. Bit 8 effectively determines 2 states or modes of operation of the processor. When the bit is zero the processor is said to be in the privileged or supervisor state, otherwise it is in non-privileged or user state.

The machine designers envisaged that (many?) user programs would run in the machine in user mode under control of some operating system or executive whose code would be executed in privileged mode. Some well protected mechanism is required to allow user programs to invoke privileged supervisor programs when requesting service from the operating system.

The mechanism for doing this, in this and several other machines, is to use a single instruction known as the supervisor call (SVC) instruction. This instruction invokes one of 16 different interrupts allowing the supervisor to be entered at one of 16 different points. The SVC is a full 32-bit instruction



The R₁ field is not used to denote any register in this instruction, but determines which of the 16 different services is being requested. The effective address (Addr + R_x) is evaluated and provided as an argument, giving further information on the service required, it is stored in memory. The memory locations involved are:-

address (hex)	role
94	: the argument is stored here.
96-99	: the old PSW (i.e. that of program which issued the SVC) is stored here.
9A	: new PSW (status part)
9C	: new PSW (LOC) for SVC 0
9E	: new PSW (LOC) for SVC 1
	..
BA	: new PSW (LOC) for SVC 15

From this it can be seen that following an SVC a given status (usually privileged) comes into effect and one of 16 different values of the location counter is loaded i.e. 16 different supervisor entry points are possible.

This method of "interrupting" into the supervisor programs is such that the user program does not need to know where the supervisor code is situated. If some form of memory protection were available to prevent user programs overwriting the SVC table area (and the supervisor programs) then a completely protected mechanism for entering supervisor would exist.

Peripheral Device Interrupts

Under normal circumstances the overwhelming majority of interrupt requests arise from peripheral (I/O) devices. Each peripheral device is connected to the processor via the multiplexor bus and a device controller. Each controller has a unique device address in the range 0-255.

Certain events taking place (instantaneously) within a device are the ultimate source of interrupt requests from the device, but it is the device controller which determines how such an event shall be treated. An interrupt arising from a peripheral device is referred to as an external interrupt.

The effect of an event in the peripheral is, almost always, to change the status of the device, its effect upon the interrupt mechanism depends on the current state of the controller. Each controller contains a 1 bit register for recording events in its device. The output of this register may, in certain states of the controller, be used to request an interrupt from the processor. Controllers may exist in one of 3 states whose names and significance is described below:-

state	meaning
disarmed	Events occurring in the device do not generate interrupt requests at all and no record is made of them in the controller.
disabled	Events in the device are recorded within the controller as generating interrupt requests, but the request is not communicated to the processor.
enabled	When events occur the request is recorded within the controller and passed to the processor. Any remaining request, recorded while the processor was disabled, will also be passed to the processor.

The enabled/disabled state of a controller is determined by sending commands to it which have the required new state coded in the 2 most significant bits of the command byte. The meanings are as follows:-

control byte	meaning
00xx xxxx	controller state remains unchanged
01xx xxxx	the controller is enabled
10xx xxxx	the controller is disabled
11xx xxxx	the controller is disarmed

Bit positions described as 'x' have no effect on the enabled/disabled state of the controller.

To summarise the description above, it can be seen that it is possible, by program, to control the interrupting capability of an individual device. Thus events occurring within the device will be seen as changing the status of the device and may also generate interrupt requests, possibly after some interval during which the device was disabled. It is not obvious from the status of a device whether or not it is currently requesting an interrupt (cp. w. DEC PDP9/15 structure).

External Interrupt

When an external interrupt occurs it is necessary for the program to be able to do 2 things viz:-

- 1) Determine which device requested the interrupt i.e. the address of the device.
- 2) Clear the interrupt request from the device.

These actions may be achieved, under program control, by means of the acknowledge interrupt (AI) instruction. The instruction exists in the RR and RX formats and its effect is detailed below:-

- 1) It clears the interrupt request from the device.
- 2) It returns the device address in the first operand register (R).
- 3) The status of the device is placed in the second operand and also in the condition code in the same manner as for a sense status instruction.

At any one time there may be several (>1) devices requesting interrupts, in which circumstance the AI instruction deals with the highest priority request. The priority of requests is the same for all requests from any one device and there is a fixed ordering among devices on a given machine, which is "wired in" with the device controllers.

The details of which specific PSW exchange takes place as the result of an external interrupt request is determined by bits 14 and 11 of the PSW status part. Bit 14 if reset to zero, disables all external interrupts. Depending on the setting of bit 11 the hardware gives more or less assistance in determining which device interrupted and despatching to the appropriate device handling code.

Basic External Interrupt (bit 14=1, bit 11=0)

In this state all external interrupt requests are granted by making a PSW exchange from location 40(hex). No other action is taken outside program control. Thus it is necessary for the program to perform an acknowledge interrupt instruction in order to determine which device is involved, and then for the program to ascertain the appropriate code to be executed.

If bit 11 is set as well as bit 14 the hardware (i.e. outside program control) performs more functions, as described below.

Immediate (External) Interrupt Mechanism (bit 14=1, bit 11=1)

The action which occurs on granting an external interrupt request when bit 11 is set is intended to assist in determining which code to despatch to when the grant occurs.

Firstly, the program is responsible for setting up and maintaining a 256 halfword table in memory locations D0-2CF (hex). This is known as the automatic i/o service table and contains one entry for each of the 256 possible devices on the machine. The entries in the table play an important role in granting external interrupt requests. In other similar machines a table such as this is often called an interrupt vector.

When an external interrupt request is granted the processor acknowledges the request, clearing it and determining the device address. It uses this device address to index into the halfword I/O service table described above. The value contained at that point in the table is taken as the address of an immediate PSW exchange area, and the PSW is exchanged. An immediate PSW exchange area is set out as shown below:-

address	use
n	: old PSW status
n+2	: old PSW location counter
n+4	: new PSW status
n+6	: the first instruction of device handling code must be in this location.

This is essentially a 3 halfword area immediately preceeding the first instruction to be obeyed after the interrupt has occurred. The term "immediate" is used to indicate that the processor rapidly, without program intervention, determines which of (possibly) 256 different device handlers to enter, and enters it.

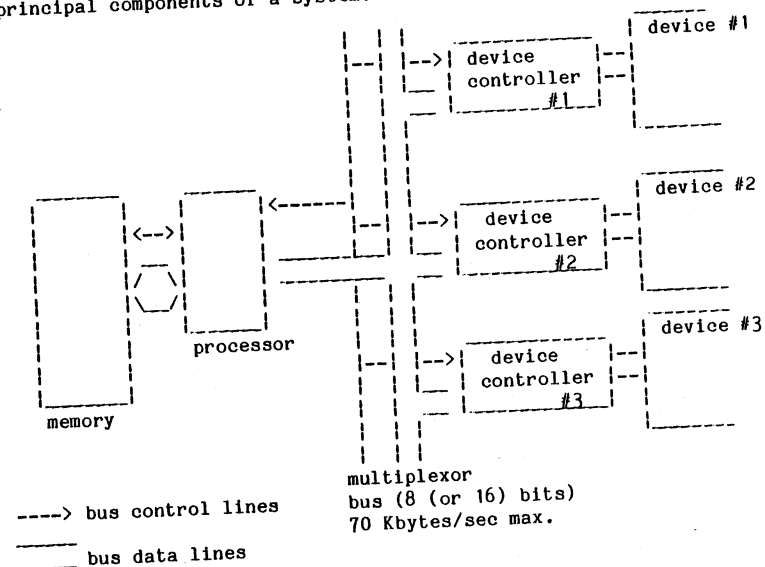
Some General Remarks about (External) Interrupts

In the first place it should be realised that in a system of any generality the processor context for any given activity will almost certainly consist of all the registers in addition to the PSW. This means that in general an interrupt invoked activity cannot make any assumptions about the use which the interrupted activity must, therefore, be prepared any register. An interrupt invoked activity must, therefore, be prepared to preserve the contents of any registers which it requires to use. Incidentally, a complete load or store multiple takes about 30 microseconds on a model 74. There is no hardware assistance with this part of a full context switch.

Related to the above observations, the immediate interrupt mechanism has some serious drawbacks in a system of any generality. At least some external interrupts lead to major changes in the state of a system (e.g. a user has completed input of line at a terminal). In this circumstance the supervisor will wish to be informed of what has happened and as a result may well wish to place some other activity on a queue for the processor, in particular, it may wish to change the activity which should be resumed next i.e. it will not be the activity which was originally interrupted. To generalise, it is not the business of a device handler to decide either where to place the context of an interrupted activity or which activity should be resumed at the conclusion of interrupt processing, yet this is the structuring indicated by the immediate interrupt mechanism.

PERIPHERAL DEVICE CONTROL

The diagram shows the paths by which information flows between the principal components of a system.



From this it can be seen that

- 1) Every device is connected to the system by a controller.
- 2) All transfers of information to and from devices pass through the processor.

Common Controller Characteristics

The purpose of the device controllers is to present each device to the processor in a standard way. The degree of standardisation that can be achieved is indicated in this section.

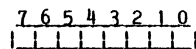
The devices themselves have some common characteristics. All are a source and/or an acceptor of a sequence of data bytes.

Via the controller the processor is able to determine at any instant whether it is possible to transfer a byte of data to or from the device.

Each controller has a unique 8 bit device address which is used by the processor to select the controller with which it wishes to communicate. The multiplexor bus is used to transfer either data or control information to or from the controller. The control information is used by the processor to determine when it is possible to transfer data.

The basic units of control information are the command byte which is transferred to the controller and the status byte which is transferred from the controller. Command bytes do things such as start, stop or alter the overall mode of operation of the device. The status byte gives information about the current state of the device. The various bits within these bytes have significances which are peculiar to each device, except that there are some common characteristics.

Status Byte



---> examine =1 => one or more of bits 4-7 set

----> busy =1 => data may not be transferred between processor and device
=0 => data, usually a single byte, may be transferred.

Command Byte

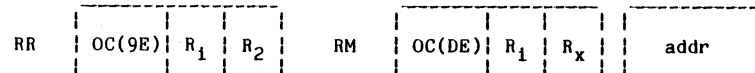
In the same manner the most significant 2 bits (7 & 6) of the control byte are invariably used to determine the interrupting capabilities of the device controller. The details are described elsewhere.

Input and Output Instructions

These instructions cause information to pass between the processor and the device controller. The specific controller to be used is always determined by the least significant byte of the register designated in the R_1 field of the instruction. The byte of information transferred is that referred to by the second operand of the instruction, either in memory or the least significant byte of the R_2 register.

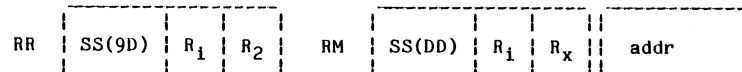
The various instructions are considered in turn, starting with 2 which transfer control information.

Output Command (OC)



The second operand is the command byte which is transferred to the controller determined by the contents of R_1 . The condition code is normally cleared.

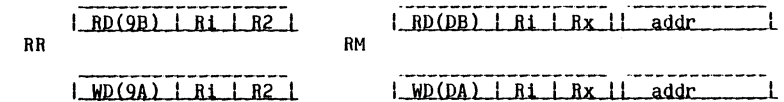
Sense Status



The status byte of the controller is transferred to the second operand. In addition the least significant 4 bits of the status byte are copied into the condition code.

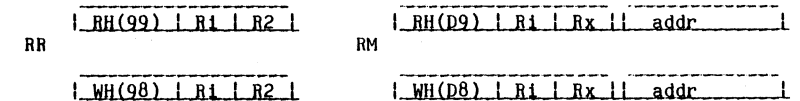
There are 3 distinct types of data transfer instruction. All of them should only be executed when the status of the device indicates that it is appropriate.

Read/Write Data (RD/WD)



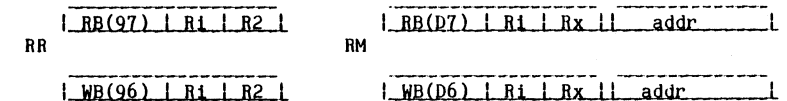
A single byte of data is either read from the device (RD) or written to it (WD). The condition code is normally cleared.

Read/Write Halfword (RH/WH)



Some devices are capable of accepting or offering 16 bit data items, these instructions would be used in this case. If the controller involved can only transfer bytes of data then the most significant byte of the data is transferred and immediately afterwards (status notwithstanding) the least significant byte is transferred. The condition code is normally cleared.

Read/Write Block (RB/WB)

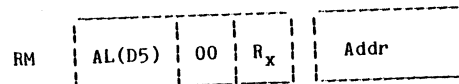


These instructions transfer a sequence of bytes between a contiguous area of memory and the device. Between each byte transferred the processor will examine the status of the device and will wait for the busy flag to clear before transferring the next byte. If any bit other than busy (bit 3) sets then the instruction is terminated prematurely. The contiguous area of memory involved is defined by its first and last byte addresses inclusively. In the RR format R_2 and R_2+1 contain these addresses. In the RM format they are contained in R_x+addr and $R_x+addr+2$ resp.

At the conclusion of the instruction the condition code will be clear if all bytes were transferred otherwise it will contain the 4 least significant bits of the status byte which terminated the instruction prematurely.

It should be emphasised that this complete activity constitutes one single (uninterruptable) instruction, whose execution could take any length of time determined by the device and controller concerned.

Auto Load (AL)



This is a unique instruction designed to enable the loading of programs into an almost completely undefined memory. The instruction assumes that locations 78 & 79 (hex) are set up with a device address and a device command resp. The R₁ field should be zero. The second (immediate) operand is used to determine which locations are to be used during the loading.

Byte 78 contains the device address of the controller to be used. First the command in byte 79 is sent to the controller. The processor then commences reading data from the device, under the control of the busy bit in its status. If any other bit in the status becomes set then the instruction terminates, setting the condition code accordingly.

An initial sequence of zero bytes is ignored. The first non-zero byte and all subsequent bytes are transferred to successive memory bytes starting at location 80(hex). The instruction terminates, normally, when a byte has been transferred into the address specified by the second operand of the AL instruction (R_x+addr).

The, so called, 50 sequence consists of an AL instruction followed by an unconditional branch to location 80. The data read is assumed to be a program, which is usually a bootstrap to load other programs.

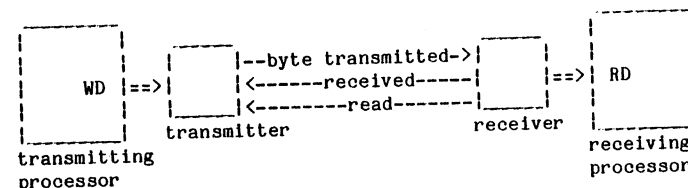
Processor Timeout

The input and output instructions assume that a valid controller address is present in R₁. If this value does not refer to an existing controller then a processor "timeout" occurs during the instruction. The processor waits for a response from the controller for a maximum of approximately 30 microseconds. If no response is forthcoming before this interval expires then the timeout occurs. The instruction is aborted and the condition code is set to the value 4 (V bit set).

There are only 4 or 5 devices on an ISIS satellite model 74. They will be considered in turn.

Inter-Processor Links

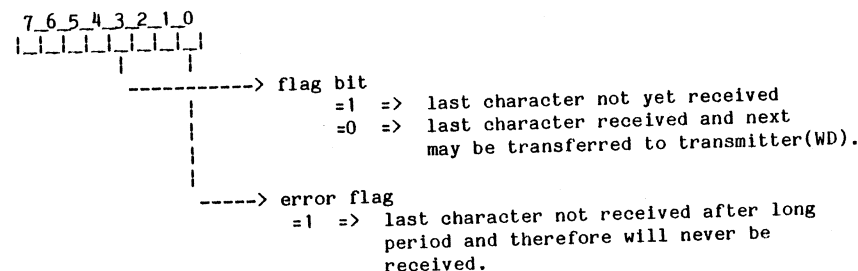
A separate handout gives a more detailed description of their operation, which is only summarised below. There are 2 distinct types of device involved: a link transmitter and a link receiver. The set up is:-



The devices and their controllers are not distinguished in the diagram. The signals pass between the transmitter along a single line (binary communication channel). "Received" means that the data has been accepted by the receiver. "Read" (past participle) means that the data has been transferred from the receiver to the receiving processor (via a RD instruction) and implies that another byte of data may now be transmitted.

The details of the programming of these devices is indicated by tabulating the meaning of the various bits in their command and status bytes.

Transmitter Status Byte



```

7 6 5 4 3 2 1 0
| | | | | | | |
| | | | | | | |
| | | | | | | |
-----> Reset i.e. assume receiver is prepared to
         receive a character and reset flag bit.
| | | | | | | |
| | | | | | | |
| | | | | | | |
-----> Clear error
         and re-transmit the last character,
         busy remains set.
| | | | | | | |
| | | | | | | |
| | | | | | | |
-----> Interrupt state control
```

```

7_6_5_4_3_2_1_0
|_|_|_|_|_|_|_|
|
----> Busy flag
      =1 => no character available
      =0 => character available

```

```

7 6 5 4 3 2 1 0
1 1 1 1 1 1 1 1
|
|
|----> Reset    The receiver is effectively switched off for
|               a period of time. At conclusion the flag is set.
|               and it is ready to receive another character
|               from the transmitter.
|
|
|-----> Next symbol
Mode
The next RD does not set the busy flag
and does not generate the "read" signal to the
transmitter. N.B. it only effects the immediately
following RD.

```

The satellite model 74's have a transmitter and receiver whose addresses are 98 and 88 resp. These are more or less permanently connected to the ISYS machine. In addition these machines have a transmitter and receiver whose addresses are 99 and 89 resp. for general use.

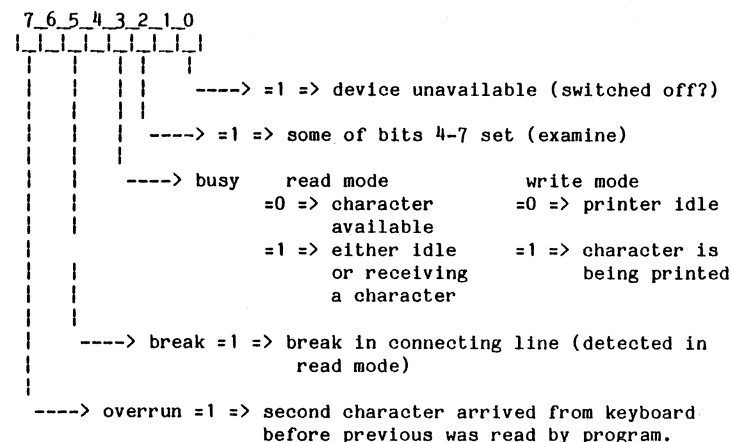
The single controller forces the terminal to be operated in a half duplex manner, that is, either to output to the printer or to take input from the keyboard, but not both concurrently, even though all terminals are capable of full duplex operation.

The actions in the event that the keyboard is struck when the controller is in write mode or an attempt is made to print a character when the controller is in read mode are left undefined!

```

7_6_5_4_3_2_1_0
|_|_|_|_|_|_|_|
| | | | | | |
|   |   |   |
|-----> 00 mode unchanged
          01 read mode
          10 write mode
          11 ?? undefined
|       |
|       |
|-----> 00 hardware echo unchanged
          10 suppress hardware echo
          01 enable hardware echo
          11 ?? undefined
|
|-----> interrupt state

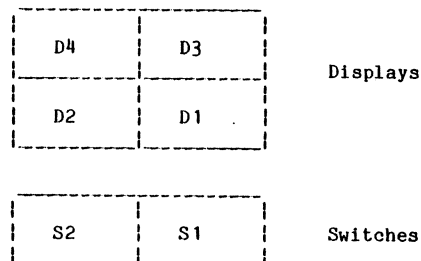
```

Status Byte:

N.B. Read/write mode, once set remains the state of the device until the next command to change it. The same is true of the hardware echo state. It is, therefore, unnecessary to set the state for every character transfer.

The Control Console (address=1)

The control console consists of a 16 bit switch register for input and two 16 bit displays for output, plus other switches and indicators. The displays and switches are designated as shown on the diagram below:-



Each byte is separately distinguished. These bytes appear as devices from a running program. Their values may be written or read by I/O instructions. However, they are fast registers and consequently may accept or offer information at processor speeds. For this reason no busy bit is included in the control console status byte, information is transferred whenever requested. The RB/WB instructions may not be used as they presuppose the meaning of the busy bit.

The mode of the controller determines which particular byte of displays or switches is transferred on a given I/O instruction. In "incremental" mode the bytes are taken in the order:- S1, S2, S1, S2 etc.. or D1, D2, D3, D4, D1, D2, D3 etc.. In "normal" mode the counter which keeps track of bytes to be transferred is reset at the start of each instruction. The bytes transferred for each instruction, in this mode, are as follows:-

RD	S1
RH	S1,S2
WD	D1
WH	D1,D2

Note that, in either mode, the least significant byte is transferred before the most significant. This is the converse of the convention for I/O instructions, which take the most significant byte of a memory word before the least. The most common programming sequences for the console are, in incremental mode:-

Switches	Displays
dev=1	dev=1
rh dev, reg	exbr reg,reg
exbr reg,reg	wh dev,reg

The exbr instructions takes the value in R_2 and exchanges the most and least significant bytes, placing the result in R_1 . The exchange is necessary in order to give the natural relation between 16 bit quantities on the console and within the memory.

The command byte for this device consists of 80 or 40(hex) to set normal or incremental mode resp.