10 Interrupts.

There is only one source of interrupts in an unexpanded machine, namely a regular time interrupt. Expansion boards may generate interrupts, but suitable software must be provided to deal with the extra interrupts.

The system runs with interrupts enabled most of the time. It is inadvisable to disable interrupts for a prolonged period if this is avoidable because the time interrupts will be missed.

A number of firmware routines enable interrupts and this is remarked upon in their descriptions. In particular the kernel routines dealing with ROMs and the restart instructions (eg. LOW JUMP) enable interrupts.

10.1 The Time Interrupt.

The time interrupt occurs roughly once every 1/300th of a second. On machines with PAL monitors (as in the UK) or SECAM monitors (as in France) the timer is synchronised with frame flyback every sixth tick. On machines using NTSC monitors (as in the US) the timer is synchronised with frame flyback every fifth tick. The time interrupt is processed by the Kernel and presented to the rest of the system in a number of ways :

a. Fast Ticker Interrupts. Period = 1/300th of a second.

For a high resolution or very short period timing (not intended for general use).

b. Sound Generation Interrupt. Period = 1/100th of a second.

This interrupt drives the sound generation firmware, but is otherwise not visible to the system.

c. Frame Flyback Interrupt. Period = 1/50th or 60th of a second.

For actions which must take place during frame flyback. Ink flashing is performed during a frame flyback interrupt, for example.

d. Ticker Interrupt. Period = 1/50th of a second.

This is the general purpose ticker interrupt. The keyboard is scanned at the start of each ticker interrupt.

e. System Clock.

There is a timer that counts fast ticks ie. 1/300ths of a second. This can be used to measure elapsed time without setting up a relatively expensive fast tick event. The timer is read by calling KL TIME PLEASE and may be set by calling KL TIME SET.

11.2 External Interrupts.

The Z80 is run in interrupt mode 1. Which is to say that all interrupts cause an RST 7 to be executed by the processor. The interrupt handling code in the Kernel can distinguish between the time interrupt and an external interrupt. It does this by re-enabling interrupts inside the interrupt routine. If the interrupt repeats then it is assumed to be an external interrupt, otherwise it is taken to be a time interrupt. Note that this requires that the source of external interrupts should not clear the interrupt condition until the software resets it.

Before an external interrupt is enabled its interrupt handler must be 'installed'. This is done by copying the 5 bytes at address #003B to a new location and replacing them by suitable code (probably including a jump). When the Kernel detects an external interrupt it calls address #003B in RAM to process the interrupt.

Entry:

No conditions.

Exit:

AF, BC, DE and HL corrupt.
All other registers preserved.

Notes:
Interrupts are disabled and must remain disabled.
The lower ROM is disabled.
The upper ROM select and state are indeterminate.
The alternate register set must not be touched.

The interrupt routine must establish whether it can deal with the interrupt, and if so it must at least clear it. If the interrupt is not the responsibility of the routine then it should jump to the copy of the bytes taken from location #003B which may be competent to deal with the interrupt. This requires the code patched at location #003B to be position independent in case a second external interrupt handler is installed. The code put at #003B at EMS is position independent - it merely returns.

Note that interrupt handling code must be in RAM somewhere between #0040 and #BFFF. Interrupt handlers should be as short as possible. If an interrupt requires a lot of processing beyond that required to clear it, then the interrupt should kick an event to do the work outside the interrupt path.

10.3 Nonmaskable Interrupts.

There is no provision for handling a nonmaskable interrupt (NMI) in the firmware (despite the fact that NMI is available on the external bus connector). Various firmware routines (notably those connected with driving the Centronics port, the PPI to access the sound chip and keyboard, and the cassette) will have timing constraints violated if NMIs occur whilst they are active. It is recommended that NMI should not be used.

10.4 Interrupts and Events.

As a general rule hardware interrupts should be transformed into their software equivalents, 'events', as soon as possible. The handling of events is more flexible than the handling of hardware interrupts – for example there are no restrictions on where event routines may reside, or on interrupt enabling.

Events are described by an event block. This block contains the event class, the event count and an event routine address. When an event occurs the event block is 'kicked' and the Kernel arranges for the event routine to be called once for each kick (the number of kicks outstanding is kept in the event block). The event routine is not necessarily called immediately. When the event routine is actually run depends on the event class as follows:

a. Express Asynchronous Events.

This is an unusual class of event. The even routine is called immediately during interrupt processing. The routine must be accessible by the interrupt code, it may not enable interrupts, corrupt the IX and IY registers or use the alternate register set. The routine should be as short as possible.

b. Normal Asynchronous Events.

This is the most flexible sort of event. When the event is kicked the event routine is not called, but the event block is placed on the interrupt event pending queue.

Once the current interrupt has been processed, just before the Kernel returns from the interrupt path, any events on the interrupt event pending queue are processed. While the events are being processed the system is running with interrupts enabled and may be regarded as no longer being in the interrupt path. It is using its own stack rather than the main system stack. This private stack is 128 bytes long.

The asynchronous event routine is, therefore, called shortly after the event is kicked and is not restricted in what it may do or where it may be located. The event routine may take as long to run as is needed. Any further kicks received during the time that the event routine is running will be added to the event count and will be processed before returning to the interrupt program.

c. Synchronous Events.

Synchronous events are queued on the synchronous event pending queue. They are not processed until the foreground program allows the queue to be processed. This can be used to control interactions between different parts of programs.

10.5 Interrupt Queues.

The various time interrupts provide three sources of 'kicks' for events. The events to be kicked when each of the interrupts occur are stored on queues, one queue for each source of kicks. The user provides an area to store for the Kernel's use. The size of the area depends on which queue it is for. The last 7 bytes of the area are always an event block which the user should initialize appropriately.

a. Fast Ticker Events.

Events on the fast ticker queue are 'kicked' on each fast ticker interrupt, i.e. every 1/300th of a second. A fast ticker block is 9 bytes long.

b. Ticker Events.

Each event on the ticker queue is associated with a timer. The timer may be a 'one shot', which goes off once, or a repeater, which goes off periodically. The timer counts ticker interrupts, i.e. 1/50ths of a second, and when sufficient have occurred it goes off. Each time the timer associated with an event goes off the event is kicked. A ticker block is 13 bytes long.

c. Frame Flyback Events.

Events on the frame flyback queue are kicked on each frame flyback interrupt, i.e. 1/50th of a second on PAL or SECAM machines and every 1/60th of a second on NTSC machines. A frame flyback block is 9 bytes long.