9 Language ROMs

The term *language ROM* is something of a misnomer given most peoples' idea of what a language is. In the context of paged ROM software the language is the primary paged ROM. Other paged ROMs may perform functions transiently but control is then returned to the current language ROM. The language ROM receives a large allocation of zero page workspace and is allocated pages 4 through to 7 as private workspace. In addition to this the language has control of the *user RAM* which may or may not be used as additional workspace. BASIC, for example, uses a variable portion of the *user RAM* (from LOMEM to HIMEM) for the storage of program variables.

Languages are most typically implemented in language ROMs as would be expected. Thus BASIC, FORTH, LISP and BCPL are all language ROMs but other software implemented as language ROMs include word processors and terminal emulators.

No paged ROM software should be executed unless a service call has been performed first with the possible exception of a language entered following a reset. The language entered after a hard reset will be the language ROM identified by the ROM type byte in its header occupying the highest priority socket. Following a soft reset the language active when the reset occurred will be reinitialised. Any language should respond to a **command* to enable its activation when this command is issued. This mechanism allows the user to switch between languages. This command would be unrecognised by the operating system which would then issue an *unrecognised * command* paged ROM service call to which the language ROM would respond via its service entry point.

9.1 Language initialisation

A language ROM will be entered via the language entry point with an accumulator value of &01 when the language is selected. The language is entered with a JMP instruction and no return is expected. The stack pointer should be reinitialised as the stack state is undefined on entry.

The language ROM should also be able to respond to service calls which may affect it (see below) e.g. be able to respond to the service call which warns of a changing OSHWM due to font explosion.

9.2 Firm keys

On the Electron the function keys are implemented as a combination key press requiring the use of the CAPS LK/FUNC key with the number keys. In addition to these soft keys there are a number of non-programmable *firm keys* which also produce text strings when pressed. The other character keys (A to Z plus the comma, full stop and slash keys) pressed in combination with the CAPS LOCK/FUNC key constitute the *firm* keys.

A language ROM indicates that it has the facility to expand these keys by bit 4 of the ROM type byte being set (see section 8.4). When the operating system detects that a firm key has been pressed it calls the language via its entry point to request the expansion of the key. The language should then yield the firm key string one character at a time in response to further calls.

The two calls made through the language entry point are:

A=2 This call expects the next key in the firm key expansion to be returned in Y.

A=3, Y=firm key code This call is an initialising call. The language should return the length of the firm key string in Y.

The key values passed to the language with this call are:

&90 to &A9	FUNC+A to FUNC+Z
&AA	FUNC+:
&AB	FUNC+;
&AC	FUNC+,
&AD	FUNC+=
&AE	FUNC+.
&AF	FUNC+/

The operating system inserts these key values into the input buffer as they are received.

OSBYTE &CC (204) may be used to read or write the OS copy of its firm key pointer and OSBYTE &CD (205) may be used to read or write the length of the current firm key string being expanded.

9.3 Language ROM compatability

It is quite feasible to write a language ROM which will work with the entire range of Acorn machines supporting paged ROMs in all their configurations.

The first question that a programmer should consider before implementing software in a Language type ROM is whether it actually needs to be a language ROM? Many utilities are only required transiently and it is better to implement them as service type ROMs. A routine in a service type ROM can then be used from the language environment.

As has been mentioned above the language should have a service entry point so that it may be selected by *a *command* and be able to respond to changes in OSHWM. For information about service type ROMs read the next chapter. It must be remembered however that a language ROM is copied across to the second processor when a Tube is active. Therefore, when executing, the language must not rely on receiving service calls (i.e. the only ones the language code should respond to are those of relevance when on an I/O processor such as the *font explosion warning*). The service code should not share or use the language work space (&400-&7FF or language zero page) because the service code is executed in the I/O processor of a Tube machine where the Tube code has the status of the current ‘language’ and the actual language is across on the second processor. The language code should not attempt to perform any manipulation of hardware by direct poking because this would make it machine dependent. The programmer may wish to implement hardware dependent routines in the service section of the ROM. The language code should communicate with the service code using *unknown OSBYTE* calls etc. for this purpose.

It is always easier to write ROM code to create software with limited compatability, It is often the case that software written originally with just one machine or configuration in mind will be just as useful on another machine. A programmer should always have confidence in his or her skills such that they consider the extra effort worthwhile. The discipline in thought required to adhere to the compatability protocols represents a professional attitude. The Electron and other Acorn products were designed by experts, and while ultimately human and thus fallible, have put great consideration into making it possible to run software over a wide a range of machines.