18 The Low Kernel Jumpblock.

The bottom of memory, from #0000 to #003F inclusive, is occupied by the code for the restart (RST) instructions and a number of Kernel entries. Most of these entries are concerned with access to subroutines in ROM and RAM. The RST's are:

> RST 0 performs a system reset.
>
> RST instructions 1 to 5 inclusive have been used to extend the Z80 instruction set to provide extra CALL and JUMP instructions, which use addresses extended to include ROM state and ROM select components.
>
> RST 6 is available to the user.
>
> RST 7 is used by interrupts.

Since all the entries supplied must be available whether the lower ROM is enabled or not, the area is copied into RAM from the ROM during power-up initialization.

The user is not intended to alter this jumpblock (except where noted in the USER RESTART and EXT INTERRUPT areas). If the user does change the area then it is the user's responsibility to ensure that the changes do not affect other programs. To some extent this can be achieved by ensuring that the lower ROM is always enabled when other programs are running. However, since the other programs may disable the lower ROM this is insufficient in most cases. Ideally the original jumpblock contents should be restored where there is any doubt.

Section 2 contains a discussion of ROMs and the memory map and section 10 contains a general discussion of external ROMs. A brief list of the routines in this area can be found in section 14.4.

LOW: RESET ENTRY **RST 0** #0000

Completely reset the machine as if powered up.

Action:

When the machine is first turned on execution starts here. Calling or jumping to #0000, or executing RST 0, resets the machine to its initial power-up state.

Entry conditions:

No conditions.

Exit conditions:

Does not return!

Notes:

All hardware is reset and the firmware is completely initialized. Once all tables and jumpblocks have been set up, control is passed to the default entry in ROM 0 (see section 10).

Related Entries.

MC START PROGRAM

LOW: LOW JUMP RST 1 #0008

Jump to lower ROM or RAM, takes inline ‘low address’ to jump to.

Action:

RST 1 is used to extend the instruction set. It is an expanded form of the jump instruction. It should be followed by a 2 byte ‘low address’ which specifies the location to jump to and the required ROM state.

Entry conditions:

All registers and flags are passed to the target routine untouched.

Exit conditions:

All registers and flags are as set by the target routine.

Notes:

The ‘low address’ following the restart instruction is laid out as follows:

Bit:	15	14	13 … 0
	U	L	Address

If the ‘U’ bit is set then the upper ROM is disabled.
If the ‘L’ bit is set then the lower ROM is disabled.
‘Address’ is the actual address of the target routine to jump to once the ROM state has been set.

When the target routine returns the ROM state is restored to what it was before the jump. To accomplish this 4 bytes are pushed onto the stack and so care should be taken when indexing up the stack (to find the address of inline parameters, for example).

The LOW JUMP, RST 1, 'instruction' may replace the first byte of a JP (jump) instruction. It is intended for use in jumpblocks. The main firmware jumpblock is made up almost exclusively of LOW JUMP ‘instructions’.

It is assumed that the destination of the jump is a routine which will return in the usual way. The restart instruction itself does not return. The value at the top of the stack when a LOW JUMP is executed must, therefore, be a return address.

Executing a LOW JUMP enables interrupts.

Related entries:

FAR CALL (RST 3)
FIRM JUMP (RST 5)
KL FAR ICALL
KL FAR PCHL,
KL LOW PCHL

LOW: KL LOW PCHL #000B

Jump to lower ROM or RAM.
Register HL contains the ‘low address’ to jump to.

Action:

Takes a ‘low address’ as a parameter and jumps to it. The ‘low address’ specifies both the address to jump to and the ROM state required.

Entry conditions:

HL contains the ‘low address’ to jump to. All registers and flags are passed to the target routine untouched.

Exit conditions:

All registers and flags are as set by the target routine.

Notes:

The ‘low address’ is laid out as follows:

Bit:	15	14	13 … 0
	U	L	Address

If the ‘U’ bit is set then the upper ROM is disabled.
If the ‘L’ bit is set then the lower ROM is disabled.
‘Address’ is the actual address to jump to, once the ROM state has been set.

When the target routine returns the ROM state is restored to what it was before the jump. To accomplish this 4 bytes are pushed onto the stack and so care should be taken when indexing up the stack (to find the address of inline parameters, for example).

It is assumed that the destination of the jump is a routine which will return in the usual way. The value at the top of the stack when a LOW PCHL is executed must, therefore, be a return address.

Interrupts are enabled.

Related entries:

KL FAR ICALL
KL FAR PCHL
LOW JUMP (RST1)
PCHL INSTRUCTION

LOW: PCBC INSTRUCTION #000E

Jump to address in BC.

Action:

Equivalent to the JP (HL) instruction (or PCHL in some assembler dialects), except that the destination is in BC not HL.

Entry conditions:

BC contains the address to jump to. All registers and flags are passed to the target routine untouched.

Exit conditions:

All registers and flags are as set by the target routine.

Notes:

Calling PCBC INSTRUCTION is a useful way of invoking a routine whose address has been picked out of a table or otherwise established at run time.

Related entries:

KL FAR PCHL
KL LOW PCHL
KL SIDE PCHL
PCDE INSTRUCTION
PCHL INSTRUCTION

LOW: SIDE CALL **RST 2** #0010

Call to a sideways ROM, takes inline ‘side address’ to call.

Action:

RST 2 is used to extend the instruction set. It is an expanded form of the CALL instruction. It should be followed by a 2 byte ‘side address’ which specifies the location to call and the required ROM selection.

Entry conditions:

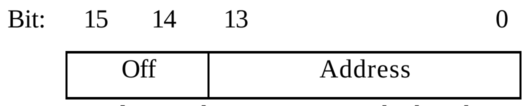
All registers and flags are passed to the target routine untouched except for IY (which is set to point at a background ROM's upper data area).

Exit conditions:

IY corrupt.
All other registers and flags are as set by the target routine.

Notes:

The ‘side address’ following the restart instruction is laid out as follows:



'Off’ gives a value in the range 0..3, which, when added to the ROM select address of the main foreground ROM, gives the ROM select address of the required ROM.

After #C000 has been added to it, ‘address’ is the address of the routine to call.

The target routine returns to the instruction immediately following the inline 'side address'. The ROM select and ROM state are restored to what they were before the call. To accomplish this 6 bytes are pushed onto the stack and so care should be taken when indexing up the stack (to find the address of inline parameters, for example).

When the target routine is entered the lower ROM is disabled and the appropriate upper ROM is selected and enabled.

SIDE CALLs are provided to support foreground programs split over a number of ROMs (up to four). See section 9 on expansion ROMs.

Interrupts are enabled.

Related entries:

FAR CALL (RST3)
KL SIDE PCHL

LOW: KL SIDE PCHL #0013

Call to a sideways ROM, HL contains the ‘side address’ to call.

Action:

Takes a ‘side address’ and calls it. The ‘side address’ specifies the address of the routine to call and which upper ROM to select.

Entry conditions:

HL contains the ‘side address’ to call.

All registers and flags are passed to the target routine untouched except for IY (which is set to point at a background ROM’s upper data area).

Exit conditions:

IY corrupt.
All other registers and flags are as set by the target routine.

Notes:

The ‘side address’ is laid out as follows:

Bit:	15	14	13	0
	Off		Address	

'Off’ gives a value in the range 0..3, which, when added to the ROM select address of the main foreground ROM, gives the ROM select address of the required ROM.

After #C000 has been added to it, ‘address’ is the address of the routine to call.

When the target routine is entered the lower ROM is disabled and the appropriate upper ROM is selected and enabled.

The target routine returns to the ROM select and ROM state are restored to what they were before the call. To accomplish this 6 bytes are pushed onto the stack and so care should be taken when indexing up the stack (to find the address of inline parameters, for example).

Side calls are provided to support foreground programs split over a number of ROMs (up to four). See section 10 on external ROMs.

Interrupts are enabled.

Related entries:

FAR CALL (RST3)
KL FAR ICALL
KL FAR PCHL

LOW: PCDE INSTRUCTION #0016

Jump to address in DE.

Action:

Equivalent to the JP (HL) instruction (or PCHL in some assembler dialects), except that the destination is in DE not HL.

Entry conditions:

DE contains the address to jump to.

All registers and flags are passed to the target routine untouched.

Exit conditions:

All registers and flags are as set by the target routine.

Notes:

Calling PCDE INSTRUCTION is a useful way of invoking a routine whose address has been picked out of a table or otherwise established at run time.

Related entries:

KL FAR PCHL
KL LOW PCHL
KL SIDE PCHL
PCBC INSTRUCTION
PCHL INSTRUCTION

LOW: FAR CALL RST 3 #0018

Call subroutine in RAM or any ROM, takes inline address of ‘far address’.

Action:

RST 3 is used to extend the instruction set. It is an expanded form of the CALL instruction that allows routines to be called anywhere in RAM or in any ROM. The restart is followed by the address of a 3 byte ‘far address’ which specifies the location to call and the required ROM state and ROM selection.

Entry conditions:

All registers and flags are passed to the target routine untouched except for IY (which is set to point at a background ROM's upper data area).

Exit conditions:

IY preserved.
All other registers and flags are as set by the target routine.

Notes:

The restart instruction takes a 2 byte inline parameter which is the address of a ‘far address’. The ‘far address’ is laid out as follows:

Byte:	0	1	2
	Address		ROM

Bytes 0..1 give the address of the routine to call.

Byte 2 is the ROM select byte which takes values as follows:

#00..#FB:	Select the given ROM, enable upper, disable lower.
#FC:	No change of ROM selection, enable upper, enable lower.
#FD:	No change of ROM selection, enable upper, disable lower.
#FE:	No change of ROM selection, disable upper, enable lower.
#FF:	No change of ROM selection, disable upper, disable lower.

The reason that the 'far address' is not contained in the FAR CALL instruction directly is because the ROM select byte for routines in ROM will depend upon the particular configuration of expansion ROMs on the machine and must therefore be established and set at run time.

Registers are passed to the target routine untouched except for the IY register. When entering a background ROM this is set to point at the base of the ROM's upper data area. (See section 10.4 and KL INIT BACK).

The target routine returns to the instruction immediately following the inline parameter. The ROM select and ROM state are restored to what they were before the call. This is accomplished by pushing values on the stack and so care should be taken when indexing up the stack after a FAR CALL instruction. (The stack usage is 4 bytes for ROM select bytes in the range #FC..# FF and 6 bytes for ROM select bytes in the range #00.. #FB.)

Interrupts are enabled.

Related entries:

KL FAR ICALL
KL FAR PCHL
LOW JUMP (RST1)
SIDE CALL (RST2)

LOW: KL FAR PCHL #001B

Call subroutine in RAM or any ROM.
C and HL contain the ‘far address’ to call.

Action:

The far call mechanism allows subroutines to be called anywhere in RAM or in any ROM. This routine takes a 'far address' and calls the given routine setting the requested ROM state and ROM selection.

Entry conditions:

HL contains the address of the routine to call.
C contains the ROM select byte.

All registers and flags are passed to the target routine untouched except for IY (which is set to point at a background ROM's upper data area).

Exit conditions:

IY preserved.
All other registers and flags are as set by the target routine.

Notes:

The ROM select byte takes values as follows:

#00..#FB:	Select the given ROM, enable upper, disable lower.
#FC:	No change of ROM selection, enable upper, enable lower.
#FD:	No change of ROM selection, enable upper, disable lower.
#FE:	No change of ROM selection, disable upper, enable lower.
#FF:	No change of ROM selection, disable upper, disable lower.

Registers are passed to the target routine untouched except for the IY index register. When entering a background ROM this is set to point at the base of the ROM's upper data area. (See section 10.4 and KL INIT BACK).

When the target routine returns, the ROM select and ROM state are restored to what they were before the call. This is accomplished by pushing values onto the stack and so care should be taken when indexing up the stack after using this routine. (The stack usage is 4 bytes for ROM select bytes in the range #FC..#FF and 6 bytes for ROM select bytes in the range #00.. #FB.)

Interrupts are enabled.

Related entries:

FAR CALL (RST3)
KL FAR ICALL
KL LOW PCHL
KL SIDE PCHL

LOW: PCHL INSTRUCTION #001E

Jump to address in HL.

Action:

Entry comprises a JP (HL) instruction (or PCHL in some assembler dialects).

Entry conditions:

HL contains the address to jump to.

All registers and flags are passed to the target routine untouched.

Exit conditions:

All registers and flags are as set by the target routine.

Notes:

Calling PCHL INSTRUCTION is a useful way of invoking a routine whose address has been picked out of a table or otherwise established at run time.

Related entries:

KL FAR PCHL
KL LOW PCHL
KL SIDE PCHL
PCBC INSTRUCTION
PCDE INSTRUCTION

LOW: RAM LAM **RST 4** #0020

LD A,(HL) with all ROMs disabled.

Action:

RST 4 is used to extend the instruction set. It is equivalent to a LD A,(HL) instruction except that it always reads from RAM irrespective of whether the ROMs are enabled or not.

Entry conditions:

HL contains the address of the location to read.

Exit conditions:

A contains the value read from the given location.

All other registers and flags preserved.

Notes:

Writing to a location always writes to RAM, even if the location is in one of the ROM areas and the ROM is enabled. The RAM LAM, RST 4, 'instruction' is the read equivalent.

Interrupts are enabled.

Related entries:

KL LDDR
KL LDIR

LOW: KL FAR ICALL #0023

Call subroutine in RAM or any ROM, HL points at ‘far address’.

Action:

The far call mechanism allows subroutines to be called anywhere in RAM or in any ROM. This routine takes the address of a 'far address' and calls the given routine setting the ROM state and ROM selection required.

Entry conditions:

HL contains the address of the ‘far address’ to call. All registers and flags are passed to the target routine untouched except for IY (which is set to point at a background ROM's upper data area).

Exit conditions:

IY preserved.

All other registers and flags are as set by the target routine.

Notes:

The parameter passed is the address of a 3 byte 'far address'. This is laid out as follows:

Byte:	0	1	2
	Address		ROM

Bytes 0..1 give the address of the routine to call.
Byte 2 is the ROM select byte which takes values as follows:

#00..#FB:	Select the given ROM, enable upper, disable lower.
#FC:	No change of ROM selection, enable upper, enable lower.
#FD:	No change of ROM selection, enable upper, disable lower.
#FE:	No change of ROM selection, disable upper, enable lower.
#FF:	No change of ROM selection, disable upper, disable lower.

Registers are passed to the target routine untouched except for the IY index register. When entering a background ROM this is set to point at the base of the ROM's upper data area. (See section 10.4 and KL INIT BACK).

When the target routine returns, the ROM select and ROM state are restored to what they were before the call. This involves pushing values onto the stack and so care should be taken in indexing up the stack after calling this routine. (The stack usage is 4 bytes for ROM select bytes in the range #FC.. #FF and 6 bytes for ROM select bytes in the range #00.. #FB.)

Interrupts are enabled.

Related entries:

KL FAR CALL
KL FAR PCHL

LOW: FIRM JUMP RST 5 #0028

Jump to lower ROM, takes inline address to jump to.

Action:

RST 5 is used to ext end the instruction set. It is an expanded form of the jump instruction for jumping to routines in the lower ROM or into the central 32K of RAM. The restart is followed by the address of the routine to jump to.

Entry conditions:

All registers and flags are passed to the target routine untouched.

Exit conditions:

All registers and flags are as set by the target routine.

Notes: The lower ROM is enabled before the jump is taken and is disabled (rather than restored) when the target routine returns. Neither the upper ROM state nor the ROM selection are changed. Two bytes are pushed onto the stack and so care should be taken when indexing up the stack (to find the address of inline parameters, for example).

It is assumed that the destination of the jump is a routine which will return in the usual way. The restart instruction itself does not return. The value at top of stack when a FIRM JUMP is executed must, therefore, be a return address.

The FIRM JUMP, RST 5, 'instruction' may replace the first byte of a JP (jump) instruction, particularly in jumpblocks, much like a LOW JUMP. A FIRM JUMP is slightly faster than a LOW JUMP but a LOW JUMP is more flexible in dealing with ROM states.

Interrupts are enabled.

Related entries:

LOW JUMP(RST1)

LOW: USER RESTART **RST 6** #0030

Undedicated RST instruction.

Action:

The eight bytes from #0030 to #0037 inclusive maybe patched as required.

Entry conditions:

Unknown.

Exit conditions:

Unknown.

Notes:

If the lower ROM is disabled when an RST 6 instruction is executed then the instructions patched into locations #0030 to #0037 are executed in the normal way.

If the lower ROM is enabled when the RST 6 instruction is executed then the firmware disables the lower ROM and jumps to #0030 to execute the instructions planted by the user.

Generally the lower ROM is disabled except while the firmware is active. Since there are no RST 6s in the firmware there should be no problem about the ROM state when a RST 6 is executed. However, to cope with all eventualities, if the lower ROM is found to be enabled when the restart is executed then the ROM state before the lower ROM is disabled is saved in location #002B. If the lower ROM is found to be disabled then location #002B is left untouched. The value stored is suitable to be passed to KL ROM RESTORE to re-enable the ROM (although KL L ROM ENABLE will have the same effect).

The user can detect whether the lower ROM was enabled when the restart was executed if location #002B is set to zero when the RST 6 area is patched and after processing each restart. If #002B is zero when the RST 6 code is entered then the lower ROM was disabled, and if it is non-zero then the lower ROM was enabled.

The default action for RST 6 as set at power-up is to perform a RST 0, i.e. a system reset.

Related entries:

None.

LOW: INTERRUPT ENTRY RST 7 #0038

Hardware interrupt entry point.

Action:

The Z80 runs in interrupt mode 1, which treats normal interrupts as RST 7 instructions. The firmware interrupt handler looks after the built in regular time interrupt. Extemal interrupts, generated by expansion hardware, are passed on to user supplied software.

Entry conditions:

No conditions.

Exit conditions:

All registers and flags preserved.

Notes:

The user must not use RST 7s as these are dedicated to the processing of interrupts. If the interrupt is from an external source then the user supplied interrupt routine, EXT INTERRUPT, is called. See section 10 for a fuller discussion of interrupts. The user may patch this area (#0038..#003A inclusive) to intercept interrupts if it is absolutely necessary (see Appendix XI, particularly section c).

Related entries:

EXT INTERRUPT

LOW: EXT INTERRUPT #003B

External interrupt routine.

Action:

The five bytes from #003B to #003F inclusive must be patched by the user if there are going to be any external interrupts. When an external interrupt is detected by the firmware interrupt handler the lower ROM is disabled and the code at #003B is called.

Entry conditions:

No conditions.

Exit conditions:

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

Notes:

When the routine is called interrupts are disabled and they must remain disabled. Under no circumstances may the user enable interrupts or use the second register set. Before the routine returns it must clear the interrupt source.

See section 11.2 for a discussion of external interrupts.

When an interrupt routine is set up the current contents of #003B..#003F should be copied elsewhere before they are replaced. If, when the routine is called, it discovers that its hardware is not responsible for the interrupt then it should jump to the copy of the previous external interrupt routine (whose hardware may be responsible).

The purpose of an interrupt routine is to clear the interrupt as quickly as possible, and perhaps perform a minimum of processing. While in the interrupt path no further interrupts are acknowledged. If the interrupt generates a substantial work load, then it should be translated into an event, so that the system is not delayed in the interrupt path for any longer than necessary (see section 11.3).

The interrupt routine must be in RAM at addresses lower than #C000 (as the ROM enable and disable routines cannot be called from the interrupt path).

The default external interrupt routine merely returns. This means that the interrupt will not be cleared and so it will repeat as soon as interrupts are re-enabled. This will cause the machine to 'lock up'.

Related entries:

INTERRUPT ENTRY
KL EVENT