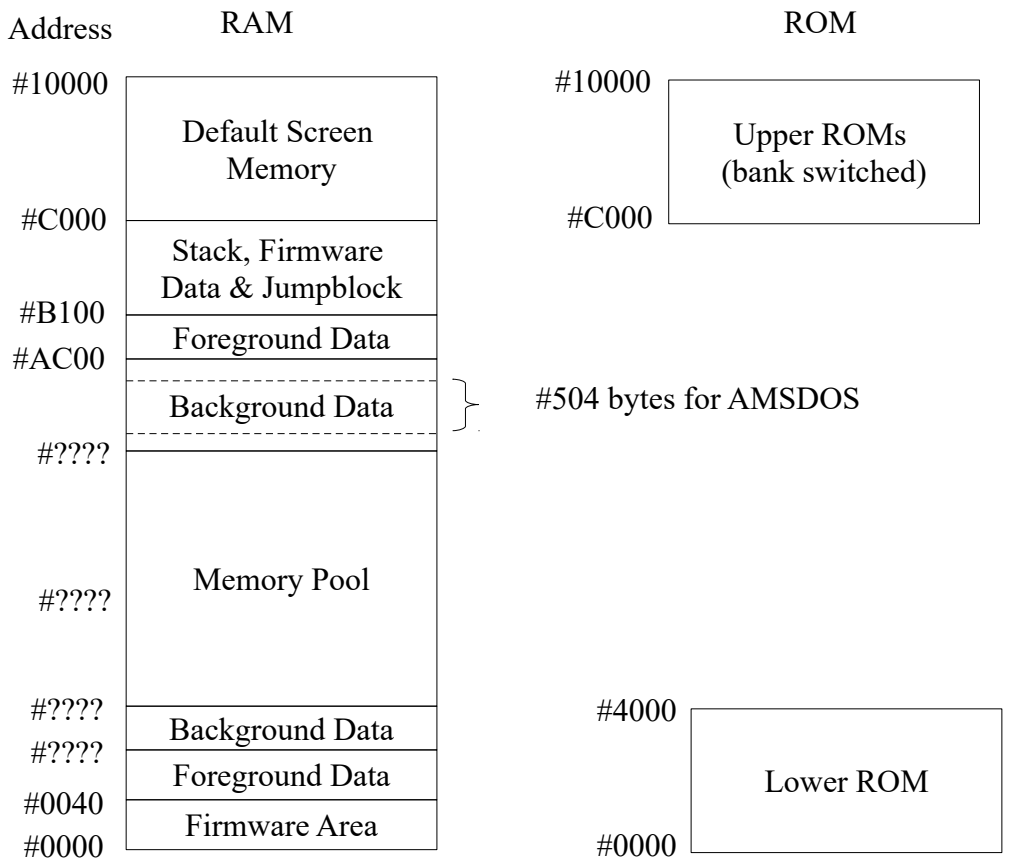
# 2 ROMs, RAM and the Restart Instructions.

The system has 32K of ROM and 64K of RAM in the Z80's 64K address space. To allow this the ROM can be enabled or disabled as required. Additional expansion ROMs can be selected giving up to 4128K of program area.

All the Z80 restart instructions, except for one, have been reserved for system use. RST 1 to RST 5 are used to extend the instruction set by implementing special call and jump instructions that enable and disable ROMs. RST 6 is available to the user.

## 2.1 Memory Map.

The memory map is complicated by the fact that into the Z80's address space of 64K bytes has been squeezed 64K bytes of RAM, 32K bytes of ROM and provision for ROM expansion of up to 252*16K (nearly 4M) bytes. The address space is divided as follows:



The sizes of the two background areas depend on the background ROMs fitted to the machine (see section 10).

The upper foreground data area need not have its lower bound at #AC00 but this is the default setting (as used by BASIC). The lower foreground data area need only be reserved if it is needed (this area is not used by BASIC and is set to zero length). The memory pool left between the background data areas is also for the foreground program to use (see section 10).

The 32K of on-board ROM is split into two sections which are handled separately. Henceforth these will be discussed as if they were separate ROMs. The firmware resides in the lower ROM. The BASIC resides in the upper ROM. This upper ROM is bank switched so that up to 252 expansion ROMs (see section 10) can replace it in the memory map.

## 2.2 ROM Selection.

There are two mechanisms for switching ROMs in and out of the address space:

**a. ROM State.**

The upper and lower ROMs may be enabled and disabled separately. When the upper ROM is enabled data read from addresses between #C000 and #FFFF is fetched from the ROM. Similarly, when the lower ROM is enabled data read from addresses between #0000 and #3FFF is fetched from the ROM. When the ROMs are disabled data is fetched from RAM.

Note that the ROM state does not affect writing which always changes the contents of RAM.

**b. ROM Select.**

Expansion ROMs are supported by switching the upper ROM area between ROMs. Expansion ROMs are addressed by a separate ROM select address byte implemented in I/O space. ROM select addresses are in the range 0..251, providing for up to 252 expansion ROMs.

When the machine is first turned on it selects ROM zero. This will usually select the on board ROM, but an expansion ROM may be fitted at this address, which will pre-empt the on-board ROM.

See section 10 for a description of the use of expansion ROMs.

## 2.3 The Restart Instructions.

The Kernel supports the store map in a number of ways. In particular a variety of facilities are provided to handle subroutine addresses extended to include ROM select and/or ROM state information. Some of the restart instructions are used to augment the existing Z80 instruction set. The other restarts are reserved.

The firmware area between #0000 and #003F is set up so that the restarts operate whatever the current ROM state is. The user should not alter the contents of this area except as indicated in section 18.

The restarts are as follows. A fuller description of their operation can be found in section 18.

**a. The Extended Instruction Set.**

LOW JUMP (RST 1)

RST 1 jumps to a routine in the lower 16K of memory. The two bytes following the restart are assumed to be a 'low address'- so RST 1 can be considered to be a three byte instruction, rather like a JP instruction.

The top 2 bits of the 'low address' define the ROM enable/disable state required; the bottom 14 bits give the actual address (in the range #0000 to #3FFF) to jump to once the ROM state is set up. When the routine returns the ROM state is restored to its original setting.

The firmware jumpblock, through which firmware routines should be called, makes extensive use of LOW JUMPs. These LOW JUMPs request the lower ROM to be enabled, so that the lower ROM may be disabled except when the firmware is active.

SIDE CALL (RST 2)

RST 2 calls a routine in an associated ROM, It has a very specialised use. A foreground program (see section 10) may require more than 16K of ROM. The side call mechanism allows for calls between two, three or four associated ROMs without reference to their actual ROM select addresses, provided that the ROMs are installed next to each other and in order.

The two bytes following the restart instruction give the 'side address' of the routine to call - so the RST 2 can be considered to be a three byte instruction, rather like a CALL instruction. The top 2 bits of the 'side address' specify which of the four ROMs to select; the bottom 14 bits, when added to #C000, give the actual routine address. The upper ROM is enabled, the lower ROM is disabled. Both the ROM state and the ROM select are restored to their original settings when the routine returns.

FAR CALL (RST 3)

RST 3 calls a routine anywhere in memory, in RAM or in any ROM. The two bytes following the restart are assumed to be the address of a 'far address'. The 'far address' is a three byte object, which takes the form:

| | |
|---|---|
| Bytes 0..1 | Actual address of routine to call. |
| Byte 2: | ROM select/state required. |

The ROM select/state byte may take the following values:

| | |
|---|---|
| 0-251: | Select the upper ROM at this ROM select address.<br>Enable the upper ROM, disable the lower ROM. |

252-255: No change of ROM select, enable/disable ROMs as follows:

252: Enable upper ROM, enable lower ROM.
253: Enable upper ROM, disable lower ROM.
254: Disable upper ROM, enable lower ROM.
255: Disable upper ROM, disable lower ROM.

Note that the 'far address' is not itself contained in the 'instruction', but is pointed at. This is because the ROM select address will depend on the particular order in which the user has chosen to install expansion ROMs and must be established at run time.

Both the ROM state and the ROM select are restored to their original settings when the routine returns.

RAM LAM (RST 4)

RST 4 reads the byte from RAM at the address given by HL. It disables both ROMs before reading and restores the state afterwards. This 'instruction' avoids the user having to put a read routine into the central 32K of RAM to access RAM hidden under a ROM.

Writing to a memory location always changes the contents of RAM whatever the ROM enable state.

FIRM JUMP (RST 5)

RST 5 turns on the lower ROM and jumps to a routine. The two bytes following the restart are assumed to be the address to jump to - so RST 5 can be considered to be a three byte instruction, rather like a JP instruction. The lower ROM is enabled before jumping to the routine and is disabled when the routine returns. The state of the upper ROM is left unchanged throughout.

**b. The Other Restarts.**

RESET (RST 0)

RST 0 resets the system as if the machine has just been turned on.

USER RESTART (RST 6)

RST 6 is available for the user. It could be used to extend the instruction set in the same way that other restarts have been used, or it could be used for another purpose such as a breakpoint instruction in a debugger.

Locations #0030 to #0037 inclusive in RAM may be patched in order to gain control of the restart. If the lower ROM is enabled when the restart is executed then the code in ROM at this address causes the current ROM state to be saved in location #002B. Then the lower ROM is disabled and the firmware jumps to location #0030 in RAM. If the lower ROM is disabled then the restart calls #0030 as normal for this Z80 restart instruction.

INTERRUPT (RST 7)

RST 7 is reserved for interrupts (see section 11), it must not be executed by a program.

## 2.4 RAM and the Firmware.

The ROM state should be transparent to the user. If the current foreground program (see section 10) is in ROM then the normal ROM state is to have the upper ROM enabled and the lower ROM disabled. If the current foreground program is in RAM then the normal state is to have both ROMs disabled. These states allow the foreground program free access to the memory pool. When a firmware routine is called the lower ROM is enabled and the upper ROM is usually disabled. This allows the firmware free access to the default screen memory (but not to all the memory pool). When the firmware routine returns the ROM state is automatically restored to what it was.

The cases where the ROM state is important are:

**a. Stack**

The hardware stack should never be below #4000, otherwise serious confusion will occur when the lower ROM is enabled and the stack is used - for example, when interrupts occur or the firmware is called.

Similarly, it is inadvisable to set the stack above #C000 unless it is certain that the upper ROM is never enabled when the stack is in use.

The system provides a stack area immediately below #C000 which is over 256 bytes long. This should be adequate for most purposes.

**b. Communication with the firmware.**

Most firmware routines take their parameters in registers. However, some use data areas in memory to pass information. Most firmware routines that use data areas in memory read these directly without using RAM LAMs (see above) or the equivalent. These routines are affected by the ROM state and the ROM select. They will read data from a ROM if the ROM is enabled and the routine is given a suitable address. (Note that the jumpblock disables the upper ROM when the firmware is called). Other firmware routines that use data areas in memory always read from RAM. They are unaffected by the ROM state and the ROM select.

Routines that always access RAM will mention this in the description of the routine. Other routines may be assumed to be affected by the ROM state. In particular the various data blocks used by the Kernel must lie in the central 32K of RAM for the Kernel to be able to use them.

**c. Communication between upper ROMs.**

Programs in upper ROMs may call routines in other ROMs, using the various Kernel facilities. There is no provision in the firmware, however, for a program in one ROM to access constants in another.

The majority of firmware routines are called via the firmware jumpblock, which starts at location #BB00, in the firmware RAM area. The Kernel routines associated with the memory map are called via one of two other jumpblock areas: the LOW area between #0000 and #003F, and the HIGH area starting at #B900. All of these routines and jumpblocks are copied out of the lower ROM into the firmware RAM area when the Kernel is initialized. Thus they work independently of the ROM state.

## 2.5 Bank Switching

The ULA in the CPC6128 includes circuitry for bank switching 128K of RAM into the 64K memory map described in section 2.1. The bank switched RAM replaces the RAM in the memory map and behaves exactly like it; in particular, it is hidden when a ROM is enabled.

The 128K of bank switched RAM is split into 8 16K blocks, numbered 0..7. There are 8 memory organizations, also numbered 0..7, each of which switches a different set of four blocks into the memory map at #0000..#3FFF, #4000..#7FFF, #8000..#BFFF and #C000..#FFFF. The user can select an organization by calling KL BANK SELECT.

The blocks available in each organization are as follows:

| Organization | Block accessed at memory address | | | |
|---|---|---|---|---|
| | #0000 | #4000 | #8000 | #C000 |
| 0 | 0 | 1 | 2 | 3 |
| 1 | 0 | 1 | 2 | 7 |
| 2 | 4 | 5 | 6 | 7 |
| 3 | 0 | 3 | 2 | 7 |
| 4 | 0 | 4 | 2 | 3 |
| 5 | 0 | 5 | 2 | 3 |
| 6 | 0 | 6 | 2 | 3 |
| 7 | 0 | 7 | 2 | 3 |

During EMS the CPC6128 selects organization 0 and this is the organization normally associated with the firmware. Note that blocks 0 and 2 contain firmware variables, firmware jumpblocks and the stack. All these need to be in their correct places for the firmware to run.

The documentation for a number of firmware routines specifies that data blocks passed to them should be in the central 32K of memory. In most cases it does not matter which blocks are switched into the memory map at these places, however, the Kernel accesses data blocks passed to it (e.g. ticker blocks or RSX command tables) at various times (e.g. during interrupts or event processing) and it has no control over the bank switching at such times. It is up to the user to ensure that the Kernel is only passed data blocks that remain accessible. The simplest solution to this problem is to ensure that all Kernel data blocks are located in block 2 (between #8000 and #BFFF).

Organizations 4..7 are the firmware organization with a new block switched into the memory map at #4000. These organizations can be used to access programs or data stored in blocks 4..7.

Organizations 1..2 are used by CP/M Plus and are not really suitable for general use. In particular, if organization 2 is selected it is necessary to patch a program into RAM at #0038 to catch interrupts and to bank switch back to a more normal organization (e.g. organization 1) to run the standard interrupt code.

Organization 3 is also used by CP/M Plus but it is of interest since it has the RAM usually used for the screen located at #4000 where it can be accessed without disabling the upper ROM.

Bank switching has no effect on the CRTC. Base addresses #0000, #4000, #8000 and #C000 correspond to the screen being in blocks 0, 1, 2 and 3 respectively. It is not possible to locate the screen in blocks 4-7. However, the firmware routines for accessing the screen memory are affected by bank switching. For example, if a base address of #C000 is set in organization 3 then the firmware will have to be told (using SCR SET POSITION) that the screen memory can be accessed at #4000; if a base address of #4000 is set in organization 3 then the firmware will be unable to access the screen memory since block 1 is not in the memory map. (See section 6.4 for a full description of the screen memory map).

Organizations 4..7 can be used to set up a complete screen in one go by using SCR SET POSITION to make the firmware write to the memory at #4000 without sending a new base address to the screen hardware. Then, when the screen has been finished, the contents of this block can be quickly copied into the block actually being used by the CRTC (using KL LDIR perhaps). For example, a title screen could be set up and bank switched out of the way and then switched back in and copied at a later date when it is wanted.