TurboMON

Sam Coupé Monitor and Toolkit

256K or 512K

Contents

Introduction

If you really hate debugging machine code programs and spend hours (or even days!) searching for the reason why things don't work, TurboMON is here to make your life easier - maybe even easy!

TurboMON is 100% self-contained machine code that handles memory paging, screens, palette, border, interrupts and the ENTIRE instruction set to allow virtually any program to be debugged using it (from simple programs to full Spectrum/Sam games/utilities and the Sam OS itself). A variety of breakpoint and execution options leave you in full control of your program at all times so you can move through your code at your own pace. TurboMON (as the name suggests) has been optimised for unbeatable performance so you can set your program running and use it as you would normally, just in slow motion. For added safety, TurboMON is write-protected from within itself so it's impossible for your code to crash it!

TurboMON is intended for people with at least a basic knowledge of Z80 machine code, but can be used as a valuable learning tool for people of all abilities. No programmer should be without the *Sam Technical Manual*, it contains almost everything you'll want/need to know about the Sam... including ports, modes and some ROM routines. *Programming the Z80* (Rodnay Zaks) is also recommended for a better understanding of the Z80 itself, and contains a tremendous amount of detail on the instruction set as well as other features such as interrupts. Together they will help you make the most of this wonderful machine.

Before getting stuck into the program itself, you are advised to read (or just skim) through the manual to give you an idea about what the manual covers - don't worry if all of it doesn't make sense first time round. I have included as much detail as I think will be of use to advanced users; beginners will begin to understand more as they learn more.

Loading TurboMON

TurboMON needs a 48K (49152 bytes, 3 RAM pages) area to run in and **must** be loaded at the start of an **odd** page number - a limitation that arises from the fact that the hardware insists mode 3&4 screens (as used for the display panel screen) must start on an EVEN page.

On a **256K** Sam this leaves the following addresses:
32768 (page 1), **65536** (page 3), **98304** (page 5), **131072** (page 7) or **163840** (page 9).

Page 11 cannot be used as it would overwrite the DOS (and MasterBASIC if you have it).
Pages 13 and 15 are unusable as there is no room before the screen and the end of memory.

On a **512K** Sam the following locations may be used in addition to the addresses above:
196608 (page 11), **229376** (page 13), **262144** (page 15), **294912** (page 17), **327680** (page 19), **360448** (page 21), **393216** (page 23) or **425984** (page 25).

Page 27 cannot be used as it would overwrite the DOS (and MasterBASIC if you have it).
Pages 29 and 31 are unusable as there is no room before the screen and the end of memory.

You will obviously want to load TurboMON where it won't overwrite your program! If you are using the 512K SC_ASSEMBLER, TurboMON can be loaded at 327680. This makes **source banks** 8 and 9 unusable (attempting to use them will corrupt TurboMON and it will need reloading), but the special markers used by the assembler are not disturbed to prevent the assembler from crashing. The last half of source bank 9 (16K starting at address 376832 (page 22) is unused and may be used for your own programs/data if you wish.

Screen Layout

Status Line

The top line of the display is the *status line* and is always shown, regardless of where you are in the program:

```
TurboMON 1.0   ROM0 ROM1 WPROT    L:31  H:01  V:30  M:4   CAPS ON
```

Some of the above text may differ, depending on what options are set:

ROM0	Shown if ROM 0 is paged in at section A (0 to 16383).
ROM1	Shown if ROM 1 is paged in at section D (49152 to 65535).
WPROT	Shown if the RAM in section A is write-protected.
L:*xx*	LMPR: Current RAM paged into block AB (0 to 32767).
H:*xx*	HMPR: Current RAM paged into block CD (32768 to 65535).
V:*xx*	VMPR: Page(s) holding monitor screen.
M:*x*	Screen mode of monitor screen.

CAPS ON Current CAPS lock status. Shift gives upper case with CAPS off and lower case when on.

The right hand side of the screen always holds the register panel. Values of all registers are shown, along with the top 8 values on the stack, indirect double register values and the flags in letters.

F register bits

The flags are shown as 8 characters, representing the 8 bits in F. Lower case letters are used if the bit is reset and upper case for set bits. Bits 3 and 5 are unused and the values of these bits should not be relied upon.

Bit 7 (Sign)	**S** for negative	**s** for positive
Bit 6 (Zero)	**Z** for zero	**z** for not zero
Bit 5 (unused)	**+** for set	**-** for reset
Bit 4 (Half Carry)	**H** for half carry	**h** for no half carry
Bit 3 (unused)	**+** for set	**-** for reset
Bit 2 (Parity/Overflow)	**E** for parity even/ overflow	**o** for parity odd/no overflow
Bit 1 (Subtract)	**N** after subtraction	**n** after addition
Bit 0 (Carry)	**C** for carry	**c** for no carry

To the right of the interrupt status (EI or DI), extra characters are used to show extra interrupt information (see the interrupts section for more information). A minus sign '-' is used to show if ALL interrupts have been disabled, a plus sign '+' if interrupts only occur at HALTs and a capital L to show if line interrupts can occur. This is also shown on the **S**tatus page.

Main Viewing Section

The main part of the screen is used for different things by different options. During the monitor mode this holds the disassembly around the current Program Counter, with a bar highlighting the next instruction to be executed. If the instruction refers to (IX+d) or (IY+d), the contents of that location are shown in square brackets after the instruction (if there is room on the line) to allow you to see the value before it is used/changed.

Other options

There are other options available to you when you are in the TurboMON's main mode, to allow you to view and edit memory. The bottom line of the display shows some of them:

```
ctrl-Quit Status Txt Num Dis Find Graph Copy Eval Mode L/H/Vmpr
```

Options are selected by pressing the uppercase letter in the required option. Press **Esc** to return from any of the options.

Leaving TurboMON

CTRL-Q: Quit TurboMON
Leave TurboMON and return to BASIC. On re-entering the system state will be exactly as you left it.

CTRL-Esc: NMI Exit
If you think you've corrupted the BASIC stack or some of the system variables from within TurboMON, you can use this exit to try to return to BASIC via BASIC's NMI routine, hopefully giving the usual *15 BREAK into program, 0:1* message. If nothing happens (he's dead Jim!) you'll probably have to reset and start again. It is advisable to use this option after using TurboMON to monitor the Sam OS, as you will have altered the BASIC stack.

System State

As well as the register panel showing the current system state, an extra page of information can be displayed:

Extra system status information: S

This option shows information not shown in the register panel and includes information that is normally stored in write-only registers but is useful to see. The following information is shown: current palette colours, interrupt handler, line interrupt port contents, full port values for LMPR/HMPR/VMPR/BORDER and current execute condition (that will be used when **CTRL-F9** is used to execute). Except for the execute options, most of the main menu options are available from the status page.

Editing Registers

The registers can be edited by pressing **EDIT** along with the register to change. e.g. **EDIT-B** to edit the contents of B.

Use **EDIT-X** for IX, **EDIT-Y** for IY, **EDIT-P** for PC and **EDIT-S** for SP.

16 bit numbers can be put in HL, DE and BC by putting the number in one of the 8 bit halves. e.g. putting 60000 in H or L will set HL to 60000. However, putting 30 in HL will require L to be set to 30 *and* H to 0.
Negative numbers are treated as 16 bit (2's complement) numbers so be careful putting negative numbers in single registers as you may change BOTH parts of double registers. Use, for example: -7&255 to mask the result to a single byte.

The following keys are used to switch to the alternate sets (remember to switch back afterwards if necessary):

CTRL-X: Perform EXX
CTRL-A : Perform EX AF,AF'
CTRL-D : Perform EX DE,HL

Quick changes to PC and SP:

Cursor_Right: Increment PC
Cursor_Left: Decrement PC

Cursor_Down: Instruction down
Cursor_Up: Instruction up

CTRL-Cursor_Up: SP = SP - 2
CTRL-Cursor_Down: SP = SP + 2

Push value onto stack: *K*

Push a 16 bit number onto the stack. Popping values off the stack is simply done by using **CTRL-Cursor_Down** to advance it by two (perhaps copying it into a register pair first - see number input for details). Pushing PC onto the stack and then giving PC a new value has the effect of CALLing a given address from the current location.

Executing Code

Code that self-modifies instructions currently shown on the screen will not appear to have worked until the instruction is highlighted (this is also the case when code pages itself out). You can force the screen to be updated using **CTRL-INV**.

Execute 1 instruction: *F7*

Execute 10 instructions: *F8*

Execute 100 instructions: *F9*

Execute (unconditionally): *CTRL-F7*

Execute as fast as possible until **CTRL-F8** is pressed. This is faster than the conditional executes since no checks are made on whether to stop. The screen is automatically switched to the monitor screen before execution.

Stop Executing: *CTRL-F8*

Stop executing and return to update panel with new register values. This facility may not respond immediately if pressed when code is executing in ROM 0, as interrupts are disabled to fetch instructions from there. This is particularly noticeable when using the command-history in MasterBASIC.

Conditional Execute: *CTRL-F9*

Execute until the current execute condition is met (see the section below for available conditions). If required, use **F3** to switch to the monitor screen before executing.

Conditional Execution

The following options allow execution to stop when a certain condition is met (the current condition is shown on the status page). **CTRL-F9** can be used to repeat with the same condition. At least one instruction is executed before the condition is tested to allow you to repeat with the same condition to find the next case. You may want to use **F3** to switch to the monitor screen before you start executing - this can be done during the number input, just before you press RETURN.

TurboMON has three automatic breakpoint conditions that are always obeyed to stop you doing silly things:
1) Stop if HALT is met with interrupts disabled (useful for setting up multiple breakpoints).
2) Stop if Program Counter reaches address zero in ROM 0 (to stop system resets).
3) Stop if LDIR about to execute with BC (block length) holding zero.

Other options available are:

Execute until PC = x: *U*

Execute until the Program Counter reaches a given address. The paging status is irrelevant since only the address has to match.

Execute until PC = x, with trace: *CTRL-T*

Same as **U** option except when the program reaches the address this option will tell you the address of instruction that diverted execution to there (usually a JP, JR, CALL or RET). Press any key to return to monitor mode.

Boundary execute, with trace: *B*

Execute until the Program Counter is outside a given boundary. When execution stops you are told the address of the instruction that caused control to be outside the boundary. Input the start and end addresses of the boundary in the range [0 to 65535] - possibly the start and end address of your code.

Execute a given number of instructions: *E*

Input how many instructions to execute [1 to 65535]. Use **CTRL-F9** to repeat same number of instructions.

Execute until a given (base) port is written to: *W*

Enter a value in the range [0 to 255], e.g. 250 for LMPR. Since the base port is used, trapping 248 will stop when any changes are made to the palette, not just palette position 0. Execution terminates *before* the OUT instruction is executed so you can see what the previous value of the port held (if applicable).

Execute until a given (base) port is read from: *R*

Details are as above option, just for reading instead.

Execute until location value changes: *X*

Execute until the contents of a given location change in value. This option notes which byte in Sam memory is currently paged in at the given address (ignoring ROMs) so that the same location is watched even if the paging status is altered. Unlike the other conditions, this option terminates *after* the location has changed and the Program Counter points to the instruction *after* the one that changed the location. You are told the previous and new value of the location.

Execute 1 instruction completely: *F4*

This option executes until the Program Counter reaches the instruction after the current one. This is particularly useful if the current instruction is a CALL - the effect is to execute the subroutine at speed and stop when the subroutine has completed and RETurns to the instruction after the one that invoked it. Most instructions will behave as with **F7** but beware of instructions that don't return control to after the current instruction (like unconditional JP, JR and RET instructions), and instructions that pop the return address to fetch data from it (some RST instructions expect data after them). This option is also very useful on the conditional jump at the end of a loop, to execute until the loop finishes.

Execute until PC = Top Of Stack: *F5*

Execute until the Program Counter matches the value currently on the top of the stack. Useful if you have started single-stepping a subroutine and you want to execute until it finishes. Be careful not to use this after values have been pushed onto the stack since the start of the subroutine, as one of these will be used instead!

Execute until the current location: *F6*

Execute until the Program Counter returns to the current location. Useful for inside program loops - will execute one more loop and stop at the same position.

Changing RAM/ROM paging

Toggle ROM 0 ON/OFF: *F0*

Toggle ROM 1 ON/OFF: *F1*

Toggle RAM Write-Protection of Section A: *F2*

When active, the above options appear in the status line as **ROM0**, **ROM1** and **WPROT** respectively.

Change LMPR page: *L*

Selects the RAM pages to be used in sections A and B (0 to 32767).

Change HMPR page: *H*

Selects the RAM pages to be used in sections C and D (32768 to 65535).

Change VMPR page: *V*

Selects the page(s) holding the display. Remember that modes 3 and 4 require the page number to be even; odd page numbers are rounded down by the hardware.

Note: The above 3 options only affect the page ONLY - other bits in the ports are unaffected. If you want to change the whole port used **CTRL-W** to send a byte to the port.

Change Screen Mode: *M*

Enter new screen mode in range [1 to 4]. This option only affects bits 5 and 6 of the VMPR, and not the page number.

Page values for LMPR, HMPR and VMPR should be in the range [0 to 15] on a 256K Sam and [0 to 31] on a 512K Sam. The current page values and screen mode are shown in the status line.

Monitor Screen

TurboMON has its own private screen for the panel to leave the normal screen free to be used by the program being run. This screen (the *monitor screen*) has its own palette, border colour and screen mode. Use **V** to change the page holding the screen and **M** to change screen mode. Unlike BASIC, when switching to mode 3 the colour of palette position 3 isn't changed to the pen colour which will make it appear *magenta* in colour. The colour of the panel screen can be changed toggled between black on white and white on black using **INV.**

Toggle between monitor screen and panel screen: *F3*

This works anytime except when executing. All TurboMON options are still available; you just can't see what's happening!

Toggle Monitor Screen ON/OFF: *CTRL-F3*

Bit 7 of the border port is set to disable the screen when in modes 3 and 4 (as well as removing RAM contention to speed the processor up to close to 6 MHz, which may be useful for testing processor intensive execution of programs).

Set palette colour: *P*

Input the position to change and then the colour for that position (the current palette is shown on the status page). Position should be in the range [0 to 15] and colour in the range [0 to 127].

Reset Palette: *CTRL-P*

This resets the monitor screen palette to the default colours used when Sam is first switched on.

Input / Output Ports

As well as your code sending values to the ports, there are two options to allow you to read from and write to them from the panel screen.

Read value from a port: *CTRL-R*

Write value to a port: *CTRL-W*

Both options expect a port value in the range [0 to 65535]. The value to send to a port is rounded to 8 bits [0 to 255]. The value read from the port is shown in the current base and in binary.

The following information shows how TurboMON deals with the Sam ports. With the exception of the SAMbus, I'm unaware of any hardware that changes memory paging - if any are built they will have to be locked out to prevent them crashing TurboMON if it pages them in.

Sam I/O port support

Read Ports:

0-223	MISC/UNUSED	Fully supported, read as normal.
31	KEMPSTON	Spectrum joystick port. Read as Sam joystick, useful for Spectrum games.
224-231	DISC 1	Fully supported.
232-239	PRINTL	Fully supported (clock uses 239).
240-247	DISC 2	Fully supported.
248	PENs	Fully supported except bit 1 (TXFMST) of LPEN always read as 0.
249	STATUS	Fully supported.
250	LMPR	Fully supported.
251	HMPR	Fully supported except bit 7 (MCNTRL) always read as 0.
252	VMPR	Fully supported except bit 7 (RXMIDI) always read as 0.
253	MIDI IN	Not supported, always read as 0.
254	KEYBOARD	Fully supported.
255	ATTRIBUTES	Fully supported (read as 0 if viewing panel screen).

Write Ports:

0-223	UNUSED(?)	Fully supported, written as normal.
224-231	DISC 1	Fully supported.
232-239	PRINTL	Fully supported (clock uses 239).
240-247	DISC 2	Fully supported.
248	CLUT	Fully supported.
249	LINE INT	Supported except interrupt doesn't occur at given line.
250	LMPR	Fully supported.
251	HMPR	Fully supported except bit 7 (MCNTRL) always sent as 0.
252	VMPR	Fully supported except bit 7 (TXMIDI) always sent as 0.
253	MIDI OUT	Not supported, nothing sent to port.
254	BORDER	Fully supported except bit 6 (THROM) always sent as 0.
255	SOUND	Fully supported.

Interrupts

TurboMON can generate *frame* and *line* interrupts but ignores the MIDI and mouse interrupts. The interrupt routine can use the status port (249) to determine which has occurred - interrupts are flagged in this port until the next EI (normally at the end of the interrupt handler), instead of lasting about 20 ms.

Single-stepping HALTs does not generate an interrupt and will just skip the HALT (unless interrupts are disabled).

Changing interrupt delay/frequency: I

The number you enter is used as the number of *real* 1/50ths of a second between interrupts, it should be in the range [1 to 254].

Entering -1 will prevent all interrupts from occurring (even if they are enabled) and entering zero will mean interrupts are only generated when a HALT is encountered (which is perfect for key routines that use HALT to wait for interrupts to read the keyboard).

If you cannot use the zero option you will need to decide how frequent you want interrupts generating. Generating interrupts too frequently can waste time and slow down the rest of your program. A value of 50 will be enough for most programs, but key inputs will require a smaller value, say 5, to speed up key responses (you can change it back once the inputting is done). For values between 1 and 254 when HALT instructions are reached the processor will wait for the next interrupt to be generated before the processor is freed; if the delay is too large time will be wasted waiting for the interrupt. Remember, give the program a chance to react in key routines, and when the same letter is required twice in succession, you may need to release the key for a second or two before pressing it again (this seems to be the case for the Spectrum ROM).

Set how many line interrupts occur per frame interrupt: CTRL-L

Enter a value in the range [0 to 255]. Since the monitor cannot easily tell when/whether line interrupts should occur, a set number of line interrupts occur for every frame interrupt (as long as the line interrupt port holds a line number less than 192).

Toggle interrupts EI/DI: CTRL-I

Generate maskable interrupt: CTRL-M

Generate an interrupt, *frame* or *line*, depending on which is due. Only works if interrupts are enabled.

Generate non-maskable interrupt: *CTRL-N*

Only works when not executing. DO NOT use SAM's own NMI button to generate NMI!

Changing interrupt mode: *CTRL-F0, CTRL-F1, CTRL-F2*

Selects interrupt mode 0, 1 and 2, respectively.

Note: Programs that use the stack to move data when interrupts are enabled (naughty!) should be run with a delay of 0 (interrupt at HALT). If interrupts are generated at the wrong time the data the stack points to will be corrupted and may cause a crash. *Starquake* and *Wizards Lair* print their sprites using a stack method and have this problem. If you have *Starquake* try typing 'PALETTE 0,0 LINE 0' before running it with the normal Sam Spectrum Emulator. This will generate a line interrupt on the top line of the display, when the game is printing its sprites, corrupting the print data addresses and will cause the game to crash! Some Spectrum Emulators on other machines are not aware of this.

Preparing to run programs

Before you can set your program running, the system must be set up as though BASIC was calling it. Most of this can be done using:

Set up system as BASIC: *CTRL-Z*

Prepare the system to run a piece of code as though it was called from BASIC. This sets up the following:

- Sets LMPR page to 31 (page in system variables), HMPR page to 1, VMPR to the screen used by BASIC.
- Switches ROM 0 on, ROM 1 off and the section A write-protection off.
- Selects interrupt mode 1, sets interrupt vector register (I) to 0 and enables interrupts.
- Selects interrupts every half a second (25), without any line interrupts.
- Sets LINE_INT port to 192 (disable line interrupts).
- Set all registers to hold zero except PC to 32768 and SP to 17500.

This leaves you to set up the following:

The Stack Pointer uses an address in the system heap area which is normally unused. BEWARE! Some commercial programs (including SC_ASSEMBLER) use this area so take care, and check the area is safe before using it. It is safest for your program to save the stack pointer and use its own private stack whilst running, and then restoring the BASIC stack. i.e.:

```
                LD    (basic_stack),SP    ; save the BASIC stack pointer
                LD    SP,basic_stack      ; set up the new stack at the end of 'new_stack'
                ...
                your code                 ; your code (or a call to it) goes here
                ...
save_stack:     LD SP,(basic_stack)       ; restore the BASIC stack pointer
                RET                       ; return to BASIC

new_stack:      DS 50                     ; 50 bytes of stack space should be plenty!
basic_stack:    DW 0                      ; store for BASIC stack
```

Memory paging set when running from BASIC

If your code is below 65536 the system doesn't have to change the paging to CALL your program and the program is already at the address you specified (set PC to this). Code above 65535 is always paged into Section C (32768 to 49151) and the paging is set up so this is true. See the *Page & Offset Conversion* section for the page number to set HMPR to. Set PC to the offset into the page + 32768. Most people prefer to assemble code to run at 32768 which keeps things simple!

Registers set when running code from BASIC

The BASIC system normally enters your code with HL and A holding certain values. If your code uses either of the values you will need to set up the registers holding the correct values before running the code.

HL holds the address of the routine being called (useful for self-relocating code). Set this to the same value used for the Program Counter (32768 + offset into page, if code is above 65536).

The BASIC CALL statement allows parameters to be passed to machine code programs. On entering the code the A register normally holds the number of parameters following the CALL statement. If your code uses parameters then CALL TurboMON with the parameters you want to use and set A to hold the number of parameters.
e.g. *CALL 32768,x,y,z,a$* then set A holding 4.

Note: Since your program has not been CALLed from anywhere, there isn't a return address on the stack to allow it to return to BASIC. When your program tries to return to BASIC it could go anywhere, corrupt programs in memory and even try to reset (albeit in slow motion!). There are a few ways to prevent this:

1) Use DI followed by HALT instead of a final RET in your program. This will force the monitor to stop at the HALT statement but will cause the program to freeze if called from BASIC.

2) If running a program with a quit option, avoid using the option!!

3) Execute until PC reaches the RET of your program that would return to BASIC. This is fiddly since reassembling may change the address.

5) (I tend to use this) PUSH zero onto the stack before executing. As long as you program (correctly) pages in ROM 0 before returning, TurboMON will try to execute the reset code at zero. This will trigger the automatic breakpoint at that address and will return to the panel screen.

Running the Sam OS

If you are interested, you can run the normal Sam system from within TurboMON! It is possible to run BASIC programs but they are very slow. To exit to BASIC: Press **CTRL-Z** to prepare the TurboMON, set PC to 102 (NMI handler), interrupt delay to 10 (to give speed up the key response a little) and execute with **CTRL-F7**! Hold the keys down until you see the response. If you directory discs, don't worry about the sector error - TurboMON runs too slowly to read properly and the DOS assumes it's an error. As you have been monitoring Sam BASIC you may have corrupted the stack so you can't return from TurboMON, so use **CTRL-Esc** instead of **CTRL-Q**.

As TurboMON is write-protected, you can also run the reset code without crashing it! However the ROM tests the RAM for faulty memory and only uses working pages ; in doing this test it thinks that TurboMON memory is faulty and only uses the RAM below it. Unfortunately, the ROM also assumes this starts at an even page, as with not having a memory expansion, and cannot cope when it fails at the odd page that TurboMON was loaded. Just a comment in case you think TurboMON can't cope and causes the problem!

Running Spectrum Software

Spectrum software has played a large part in testing TurboMON and you can monitor snapshots to see how your favourite games work or to hack them for infinite lives etc. TurboMON even runs some games that fail with PC Spectrum Emulators! The snapshots must be in .SNA format (used by some PC Emulators) or can be converted from SNP 48K by a program on the main disc (called "SNP_2_SNA") - although it currently requires MasterDOS.

Ideally you should have a copy of the Spectrum ROM, but the skeleton ROM used by the Sam Spectrum emulator should be enough for most games. If you are using the skeleton ROM and have problems running the snapshots, don't blame TurboMON since it is more likely to be the ROM!

Load the (Spectrum) ROM at 65536 (page 3) using: *LOAD "SpecROM"CODE 65536*
Load the .SNA file using: *LOAD "GAME.SNA"CODE 81893*
Load TurboMON at 131072 (page 7) and enter it.

Loading the snapshot overwrites the last 27 bytes of the ROM image with the register values, but since this area only contains the last few letters of the character set (including the copyright symbol) it shouldn't matter.

Press **EDIT-Z** to set up the system for the snapshot and load the registers from the snapshot image. The register panel will now hold the snapshot register values. Pressing **F3** should view the snapshot screen.

You are now free to execute it, hack or whatever... enjoy!

Spectrum Software Hints:

With the exception of 128K sound, the Spectrum produces sound by processor intensive toggling of the BEEP bit in the border port (254/#FE), slowing execution down considerably. You hear the slow loops as fast clicking sound. Eliminating the large delay loops between toggles will speed things up. Use the **W** option to watch writes to port 254 and then use your own judgement on which instructions to delete (probably a JR NZ,d or DJNZ e instruction but sound routines differ so this is only a guide). **EDIT-DELETE** will delete the instruction highlighted in monitor mode.

You may need to hold keys down for a couple of seconds for the program to register them, maybe increasing the interrupt frequency first. The hardest thing about running snapshots is deciding which interrupt option to use!

If the screen remains blank for a while in games, it may be drawing the game screen under *black on black* attributes or building screens/levels in memory so give the program a chance! It is fun to fill the attributes with a value that enables you to watch it as it draws - try filling from 22528, length 768, with 71 with some games.

Other Options

Text Editing: *T*

Enter the address of the text to be edited or just press return for the Program Counter address.

Text is displayed in rows of 32 characters ; out of range characters are displayed as a dot. The bottom line of the screen shows more information about the location of the cursor including the PEEK of the location in the current base, binary, and its ASCII representation (or the BASIC keyword it represents, not including ones added by MasterDOS etc.).

Keys:

A (command mode only):	Input new address.
D (command mode only):	Switch to disassembler at cursor's location.
N (command mode only):	Switch to number mode at cursor's location.

Cursor Keys:	Move cursor about text area of screen.
SHIFT-Cursor_UP	Page up.
SHIFT-Cursor_DOWN	Page down.
RETURN	Toggle command/overtype mode.

Text mode starts in command mode allowing most of the normal options to be accessed.
Pressing **RETURN** you will enter overtype mode. In this mode any text typed will be poked into memory at the cursor location. When you have finished your editing press **RETURN** to return to command mode (this will stop you corrupting code by accidentally typing text over it). The current mode is shown on the bottom line of the screen.

Typing text over the ROMs, write-protected RAM or TurboMON's private memory will have no effect and the cursor will be advanced without making any changes. You can use **F2** to remove the write-protection of RAM section A.

Number Editing: *N*

Enter the address of the numbers to edit or press return for the Program Counter address.

8 numbers (in the current base) are displayed per row and the bottom line contains information about the location of the cursor (as with the text option) ; the BASIC keyword is not shown to leave space for number inputs.

Keys:

A:	Input new address.
D:	Switch to disassembler at cursor's location.
T:	Switch to text mode at cursor's location.
Cursor Keys:	Move cursor about numbers area of screen.
SHIFT-Cursor_UP	Page up.
SHIFT-Cursor_DOWN	Page down.
RETURN	Input number(s) at cursor location / finish inputting numbers.

Pressing **RETURN** gives an input prompt. Numbers entered will be poked into the location of the cursor and the cursor will be advanced. 16 bit numbers can also be input and will be poked into 2 locations. An empty input or pressing **Esc** will return to the number mode.

As with the text option, write-protected areas cannot be changed.

Disassembler: *D*

Enter address of code to disassemble or press return for the Program Counter address.

Almost the same as normal monitor disassembly except you can disassemble other areas of memory without changing the Program Counter value - it is the easy way to browse through code.

Keys:

A:	Input new address.
N:	Switch to number mode at cursor's location.
T:	Switch to text mode at cursor's location.
SPACE:	Save current address.
SHIFT-SPACE:	Restore saved address.
Cursor_UP:	Up 1 instruction (approximate, but usually right.).
Cursor_DOWN:	Down 1 instruction.
SHIFT-Cursor_UP:	Page up (approx.).
SHIFT-Cursor_DOWN:	Page down.

Pressing **SPACE** will save the current address so you can look at another address (possibly following CALL addresses) without having to remember the original address. Press **SHIFT-SPACE** to return to the saved address.

Block Fill: *CTRL-F*

This option allows you to fill areas of (paged in) memory with a given byte. It is useful for clearing the screen or memory areas and can be used to change the screen attributes of Spectrum games so you can see game screens being drawn.

Enter the start address of the block to fill, the length of the block and the byte to fill with. The option is aborted if any of the inputs are blank or the block length is input as zero.

Useful block lengths are:

24576	(24K)	Mode 3 and 4 screen length.
6144	(6K)	Modes 1 (Spectrum mode) and 2 data length.
768	(0.75K)	Mode 1 attributes length.

If running Spectrum software as described earlier, the screen data starts at 16384 and the attributes start at 22528.

Block Copy: *C*

This option allows you to copy areas of monitor memory to another location in monitor memory. By changing the LMPR and HMPR values you can copy 32K blocks from anywhere to anywhere in Sam memory - larger blocks will have to be done in sections. The copy is intelligent to allow the source and target areas to overlap, but copying large areas between 32K sections when Block AB and Block CD hold a common page will cause problems! i.e. LMPR=1 and HMPR=2 will mean page 2 exists at 16384 as well as 32768 ... TurboMON assumes the 64K address range holds unique locations.

Enter the source address, the target address and the length of block to copy. The option is aborted if any of the inputs are blank or the block length is input as zero.

Graphs of memory areas: *G*

This option allows you to *see* where programs are and how big they are ; it is useful to *see* which areas of memory are being used and how much space you have left.

Enter the page number to display in the range [0 to 15] for a 256K Sam or [0 to 31] for a 512K Sam.

When viewing a single page, each vertical line represents 47 bytes in that page. The length of the line depends how many of the 47 bytes are not zero. The striped scale below the graph allows you to guess how big

programs or free areas are, the amount of memory each scale unit represents is shown on the bottom line of the screen along with the number of bytes that were zero.

There are 3 other ranges you can show as a graph. An empty input will give a graph covering the 64K address range currently paged in (including ROMs if paged in). Entering *256* will show pages [0 to 15] and entering *512* will show [0 to 31]. Any other values will be rounded to 5 bits (0 to 31) and that page will be shown. Pages 16 to 31 on a 256K Sam are meaningless.

As a general rule, you can usually spot the difference between machine code and data just by looking at the graph! Machine code contains few zeros and most of the peaks will be close to the top of the screen. Graphical data (except compressed data) normally contains many zeros and most peaks will only reach about half way up. All graphs show enough detail to be able to spot a single byte in the middle of a clear area!

Memory Search: *F*

Search a given page or range for sets of numbers (or characters by using " - see *Input of Numbers* section for details).
Input the page number to search or, *256* to search pages [0 to 15], *512* to search pages [0 to 31] or give an empty input to search the 64K address range currently paged in (including ROMs if paged in).

Input up to 10 separate numbers/characters (16 bit values count as 2 numbers) to search for and give an empty input to start the search. The search will start automatically on inputting the 10th value. The search also allows *wildcard* values which match any byte; input an asterisk (*) instead of a number. e.g. 33, *, *, 53 will search for LD HL,x DEC (HL), where x matches any address. This facility can be particularly useful for finding 'pokes' for games (the above example may be a method used to decrement the number of lives/shots etc.).

If you were searching the 64K currently paged in and a match is found, you are given the address in memory where it was found. If you were searching a page (or a range), you are given the page number [0 to 31] and the offset into the page [0 to 16383] where the match was found.

When matches are found you are shown 3 bytes before the matching location (in green/red), and 6 bytes after the location (in black/white).

Evaluate Expression: CTRL-E

This option will show you all the different things a number can represent: Decimal, Hex, Binary, ASCII and 2's complement (signed). These variations are shown for the whole number as well as the High and Low parts. The number input allows you to input expressions so you can use it instead of having a calculator handy.

ADDITIONAL INFORMATION:

Input of Numbers

Although the base in which numbers are displayed on the screen can be altered, the format for the input of numbers remains the same, regardless of the base settings.

If an empty input is given, the default value for that particular input is used ; if there is no default, the option is aborted (**Esc**ape can be used at any time to abort the input).

Expression Format

Hex numbers need to be prefixed with a hash (**#**), binary numbers with a percentage sign (**%**) and ASCII characters with a double quote ("), but don't use a quote after the character.

PEEK of a memory location can be obtained by surrounding a number or expression with square brackets (**[** and **]**), and DPEEK by using curly brackets (**{** and **}**). Round brackets are used to force priority (as normal) in expressions.

Registers can be referred to like variables, valid ones are: **A**, **B**, **C**, **D**, **E**, **H**, **L**, **BC**, **DE**, **HL**, **IX**, **IY**, **SP**, **PC**, **I**, **R** and **TOS** (value on top of stack). They are only recognised if typed in upper-case.

The following operators may also be used: + (add), - (subtract and negate), * (multiply), / (divide or DIV), ! (arithmetic OR), & (arithmetic AND), @ (arithmetic XOR), ? (MOD).

Since we are using integer arithmetic, multiply has slightly higher priority than divide. This helps to avoid rounding errors by multiplying before dividing. When using brackets, try to ensure *you* get it right!

Syntax Errors or Division by zero

If the input line contains syntax errors or you are trying to divide by zero, the screen will flash red and leave the input line to be corrected.

Useful expressions

(IX+d) can be obtained by:	**[IX+**d**]** where d is a specified.
IXh and IXl can be obtained by:	**IX/256** and **IX?256** respectively.
Address of Interrupt mode 2 handler:	**{I*256}**
Form address from MSB and LSB:	MSB***256+**LSB where MSB & LSB are specified.

Debugging Tips

So, your latest masterpiece crashes when you run it... what next?
This section contains a few suggestions on how to tackle your buggy programs using TurboMON.

Check your code *before* you use TurboMON to debug it - prevention of bugs is better than using TurboMON to cure them afterwards. Make full use of single-stepping programs - you'll be surprised at the number of things you can overlook... even with a relatively simple piece of code.

TurboMON is fully write-protected from within itself, but your program may destroy the BASIC system so you can't return to BASIC; in which case you are safe until you try to return... use **CTRL-Esc** to *try* to exit.

It is possible that your program never starts running - code that moves itself around (possibly paging itself in at zero) should be single-stepped to see that it is works correctly. One mistake that people often make is to forget that bit 5 of LMPR is reset to page ROM 0 in - so writing 5 to LMPR will page in the ROM as well as

changing the RAM underneath to page 5. You will also need to check that your code restores the system paging/stack after execution - execute until the final RET of your code and check the page values, ROM 0/1 status and interrupt mode etc.

If you are using interrupts, try running with all interrupts disabled (interrupt delay of -1). If the program no longer crashes the problem may be in the interrupt handler. If you are running in mode 2 interrupts check the vector table is 257 bytes long and that it points to a valid location (holding the interrupt code or a JP to it). Execute until the start of the handler and single-step through it. If this still doesn't help, check you have enough stack space to push the registers at the start of the interrupt handler, bearing in mind an interrupt could occur when other things may have already been pushed onto the stack. This is a bitch to track down!

If your code resets without (apparently) corrupting any memory it usually arises from executing code at zero (or around zero) in ROM 0. TurboMON has an automatic breakpoint at that location so try executing your program without setting any breakpoints. If execution stops at this point you can then re-run using the **CTRL-T** trace option to find the address of the instruction that caused the Program Counter to hold zero ; you may have RETurned when data was on the stack or paged out the stack before RETurning! Check your PUSHes and POPs to make sure they are balanced and you are not over/under POPping the stack.

If your code is corrupting memory and you know which area of memory is being corrupted use the **X** option to watch the location so you can see which instruction changed the location. If this forms part of your code you can re-run the program and execute until the start of that section of code. At this point you can single-step through it and see if it's doing what it should be! If the subroutine is called multiple times you can use **CTRL-F9** to execute until the next time it is called etc.

If your code freezes when run normally, run with TurboMON and use **CTRL-F8** to stop execution when you think the program is 'stuck'. Hopefully the reason why it is stuck (infinite loop?) should be easy to see. As well as endless loops you may be encountering a HALT with interrupts disabled - TurboMON automatically stops if it happens when executing.

If your code is completely self-contained (making no CALLs to the ROMs or code not part of itself) but the program resets when run, use the **B** option and input the start and end addresses of your code as the boundary. As soon as anything outside the boundary is about to be executed, PC will hold the address where it went to and you are told the address of where it came from.

Reading the Keyboard

This section is contains a useful table on how to read individual keys on the Sam keyboard. It is not shown in the technical manual and most people (myself included) end up drawing their own table - I thought it would be nice to have one in print for a change:

			Base Port							
			249/F9 (status)			254/FE (keyboard)				
254	#FE	%11111110	F3	F2	F1	V	C	X	Z	Shift
253	#FD	%11111101	F6	F5	F4	G	F	D	S	A
251	#FB	%11111011	F9	F8	F7	T	R	E	W	Q
247	#F7	%11110111	Caps	Tab	Esc	5	4	3	2	1
239	#EF	%11101111	DEL	+	-	6	7	8	9	0
223	#DF	%11011111	F0	"	=	Y	U	I	O	P
191	#BF	%10111111	Edit	:	;	H	J	K	L	Return
127	#7F	%01111111	Inv	.	,	B	N	M	Sym	Space
255	#FF	%11111111	-	-	-	Right	Left	Down	Up	Cntrl
Key bit positions:			7	6	5	4	3	2	1	0

Two ports hold the keyboard information: 254/FE holds 5 bits of information (bits 0 to 4) and 249/F9 holds 3 bits (bits 5 to 7, mainly for new keys the Spectrum didn't have). The address line is used to select which row of keys to read.

Reading keys using IN A,(n):

- Load A with the address line value for the row containing the key.
- Read from the port managing the key.
- Test the bit containing the key (the bit is reset if the key is being pressed).

Example, reading 'U':

```
LD   A,223
IN   A,(254)
AND  %00001000
JP   Z,U_pressed
```

Example, reading 'Esc':

```
LD   A,247
IN   A,(249)
AND  %00100000
JP   Z,Esc_pressed
```

When reading keys using IN r,(C), load B with the address line value and C with the port.

Example, waiting for any key to be pressed:

```
wait_for_key:XOR  A              ; all key lines are to be read
             IN   A,(254)        ; read from KEYBOARD port
             AND  %00011111      ; strip off the 5 bits containing key information
             LD   B,A            ; save for later
             XOR  A              ; all key lines to be read
             IN   A,(249)        ; read from STATUS port
             AND  %11100000      ; mask off the 3 bits containing key information
             OR   B              ; combine with the keys read earlier giving 255 if nothing pressed
             INC  A              ; increment 255 round to zero (setting zero flag) if that was the case
             JR   Z,wait_for_key ; jump back to re-read keys if nothing was pressed.
```

I'll stop there, before it turns into a full Z80 lesson!

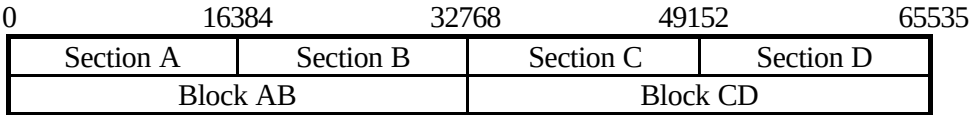
Sam Memory System

Understanding how the memory works on the Sam is fairly fundamental when programming in machine code, so if you are at all unsure read on!

The addressing system used by BASIC is just a convenient way of handling lots of memory. The Z80 can only address 64K of memory at a time and the Sam has to swap sections of it in order to access more. All addresses have a value from 0 to 65535. The position in Sam memory that this represents is dependant on the LMPR (Low Memory Page Register) and HMPR (High Memory Page Register) values, which are altered using the **L** and **H** options respectively.

The PAGING system

To picture the Sam's paging system it's best to visualise the 64k addressing range as 2 blocks of 2 sections of 16k (1 page = 16384 bytes), represented by the letters AB and CD:



Block AB is managed by the LMPR (port 250/#FA).
Block CD is managed by the HMPR (port 251/#FB).

If, using the **L** option, we change LMPR to 1 - Section A of the memory will hold page 1. Section B is always allocated 1 page above Section A, in this case to page 2.
If, using the **H** option, we change HMPR to 31 then Section C of the memory will hold page 31. Section D is allocated 1 page above Section C, in this case to page 0.

Since 5 bits are used to store the page number, 32 cannot be stored. This causes page 0 to come after page 31. On a 256K only pages 0 to 15 (inclusive) hold available memory ; 16 to 31 are unusable even though you can page them in. 256K. LMPR is 31 in BASIC to allow the system variables (page 0) to be in Section B.

Note: Remember that the other bits in LMPR and HMPR are used for other things and they may need combining with the page number before writing them to the ports.

The ROMs

When ROM 0 is paged in, it covers Section A, without affecting Section B.
When ROM 1 is paged in, it covers Section D, without affecting Section D.

Page & Offset conversions

In machine code, memory locations are normally referred to using the RAM page [0 to 15, or 0 to 31 for 512K Sams] and the offset into the page [0 to 16383]. You may need to convert from a BASIC address to this format; use the list below and look for the *largest* address *below* the BASIC address - this will give you the page number. Subtract this address from the original address to give you the offset (you can use the number input for this since the result is less than 65536).

Page 0:	16384	Page 8:	147456	Page 16:	278528	Page 24:	409600
Page 1:	32768	Page 9:	163840	Page 17:	294912	Page 25:	425984
Page 2:	49152	Page 10:	180224	Page 18:	311296	Page 26:	442368
Page 3:	65536	Page 11:	196608	Page 19:	327680	Page 27:	458752
Page 4:	81920	Page 12:	212992	Page 20:	344064	Page 28:	475136
Page 5:	98304	Page 13:	229376	Page 21:	360448	Page 29:	491520
Page 6:	114688	Page 14:	245760	Page 22:	376832	Page 30:	507904
Page 7:	131072	Page 15:	262144	Page 23:	393216	Page 31:	524288

e.g. converting 123456 to page & offset:

114688 is the *largest* address *below* 123456 giving RAM page 6.
Subtract this from the original address: 123456 - 114688 gives use the offset as 8768.

Execution Optimisations

On top of the highly optimised self-modifying code that forms the core of TurboMON, further speed is achieved by some instructions watching for repeated instructions or recognising patterns of instructions:

LDI watches for repeated LDIs and will perform up to a block of 4 at a time, as long as the block's source and target areas do not overlap. Multiple LDIs are used for copying data (many games use it and the ROM uses it for scrolling the screen). In reality a stack method is the fastest way to move data (about 2 T-states per byte faster than other methods), then using LDIs and then with LDIR. With TurboMON LDIR is the fastest due to its low fetch/decode overhead compared to both of the other methods.

LDIR has had special attention as it's used for copying *and* filling. If DE=HL+1 it is being used for filling and will be dealt with separately and quickly. Execution is also fast if the source and target blocks do not overlap - if they do the data is copied byte by byte which can be slow but is necessary to perform the same operation as the real LDIR would. LDIR is used extensively in Spectrum Manic Miner and Jet Set Willy, which both benefit greatly from the speed-up, allowing them to run at about 2 frames per second. LDDR has no such optimisation (yet?) so is slower, but is not so frequently used.

PUSH HL/DE look for repeated instructions up to a block of 3 at a time. Multiple PUSHes are used for clearing (or filling) areas of memory and this method is the fastest possible (in reality).
PUSH BC checks for PUSH DE (and then PUSH HL). Most people seem to PUSH registers in that order so we may as well make use of it! This will speed up stack copying methods that push registers in that order.
POP HL checks for POP DE (and then POP BC). This complements the PUSH checks.
PUSH AF checks for PUSH DE and then maybe PUSH BC (these 3 are used by a few games), and also POP BC checks for POP DE and then maybe POP AF, to complement the PUSH checks.
The best speed up is gained when the 3 instructions follow each-other but some time is saved if only 2 of them are present.

TurboMON recognises some delays (mainly used in Spectrum sound/tape routines) and will short-circuit them and fool the program to think they were executed fully. The following delay methods are recognised:

```
    delay1: DJNZ delay1
```

and

```
    delay2: DEC  r          where r = B,C,D,E,H,L,A  (not (HL) or index register halves)
            JR   NZ,delay2 or JP  NZ,delay2
```

Bigger delays using double registers are normally present for a reason and are executed in full (this is the case for the delay in the Sam ROM reset code, which is useful to delay the memory being cleared). They are slightly speeded up because the OR instruction checks for a trailing JR NZ or JP NZ. You can always skip the loops using **Cursor_DOWN** or delete the looping instruction using **EDIT-DELETE** (if in RAM).

There are 2 optimisations which speed up the Spectrum ROM reset code so a reset takes 34 seconds (not bad considering the amount of work it has to do!). There is another optimisation which shortens the BEEP routine, as used by the key-click, so you may use the ROM as normal. These are all present without needing to alter the Spectrum ROM image.

Note: The above optimisations are **not** performed when single-stepping or executing 10 & 100 instructions (to avoid confusion). The execute condition (if any) is not checked in the middle of a block of LDIs, PUSHes/POPs or directly after a DEC r, for speed reasons. This is unlikely to cause problems but you should be aware of it.

Limitations

The main limitations arise from real time interaction. TurboMON obviously can't be used to run time critical code (e.g. loading and saving) and expect it to work as normal! Neither can it expect to trigger line interrupts at the correct lines, but the options provided will cater for most testing needs. Programs relying on the HPEN/LPEN values (for screen refresh position) are unlikely to work as planned, but since the values are read from the normal ports, they will work when the screen refresh happens to be in the right place.

At present, the refresh register is not changed after every instruction - this would require quite a big overhead to each instruction, slowing down execution considerably. The value is changed whenever it is read from, and is set from the real refresh register. Some (mainly Spectrum) games use the refresh register to decrypt program code (mainly just the loading code) and TurboMON will not do this correctly. The refresh register on

the Sam is handled differently from the Spectrum, so the Sam may have the same problems when running them normally!

Other limitations arise from me not being able to afford the hardware to test with:
The mouse interrupt is unlikely to be featured in future versions (even if I do get one) because of the number of interrupts it would need to generate...

With the SAMbus, the clock port will work as normal but the extra memory expansion cannot be accessed from within TurboMON. This may be implemented if I get one.

MIDI may be implemented when I get more details on the requirements (especially since interrupts are involved).

And Finally the Weather...

TurboMON started life as an interesting idea in my final year at University (not part of my degree though!) and has developed into something bigger and better than I ever imagined! I hope it reflects the amount of time and effort I have put into it, and takes some of the pain out of your debugging.

Thanks must go to my fiancée, Alyson, for enduring me talking about TurboMON quite a lot!

Greetings to Paul 'Peebs' Wardle (you can have the source now!), Frode 'The President' Tennebø (hope the HHGTTG tapes were good) and Matthew 'utter genius' Collier.

Comments, ideas etc. (positive and negative) are more than welcome…

Simon Owen, January 1994

Keyboard Summary

All the options below are available in the main monitor mode but some (mainly execution related) are not available from other options.

Esc: Leave current option/input.
INV: Invert panel screen colours.

A: -
B: Boundary execute, with trace.
C: Copy memory block.
D: Disassembler.
E: Execute a set number instructions.
F: Find numbers/characters.
G: Graph of memory pages.
H: Change HMPR page.
I: Change interrupt delay.
J: -
K: Push value onto stack.
L: Change LMPR page.
M: Change monitor screen mode.
N: View and edit memory as numbers.
O: -
P: Set palette colours.
Q: -
R: Execute until a port is read from.
S: Status page.
T: View and edit memory as ASCII text.
U: Execute until PC = x.
V: Change VMPR page.
W: Execute until a port is written to.
X: Execute until a location changes.
Y: -
Z: Execute until instruction reached.

F0: Toggle ROM 0 ON/OFF.
F1: Toggle ROM 1 ON/OFF.
F2: Toggle section A write-protection
F3: Toggle to monitor screen.
F4: Execute 1 instruction completely.
F5: Execute until PC = Top Of Stack
F6: Execute until the current position.
F7: Execute 1 instruction.
F8: Execute 10 instructions.
F9: Execute 100 instructions.

Cursor_UP: Instruction up.
Cursor_DOWN: Instruction down.
Cursor_LEFT: PC = PC - 1
Cursor_RIGHT: PC = PC + 1

CTRL-Esc: NMI Exit from TurboMON.
CTRL-INV: Refresh screen.

CTRL-A: Perform EX AF,AF'.
CTRL-B: Toggle base Decimal/Hex.
CTRL-C: -
CTRL-D: Perform EX DE,HL.
CTRL-E: Evaluate expressions to different formats.
CTRL-F: Fill memory with a given byte.
CTRL-G: -
CTRL-H: -
CTRL-I: Toggle interrupts EI/DI.
CTRL-J: -
CTRL-K: -
CTRL-L: Change line interrupts per frame interrupt.
CTRL-M: Generate maskable interrupt.
CTRL-N: Generate non-maskable interrupt.
CTRL-O: -
CTRL-P: Reset monitor palette.
CTRL-Q: Quit monitor.
CTRL-R: Read a value from a port.
CTRL-S: Toggle sound chip ON/OFF.
CTRL-T: Execute until PC = x, with trace.
CTRL-U: -
CTRL-V: -
CTRL-W: Send a value to a port.
CTRL-X: Perform EXX.
CTRL-Y: -
CTRL-Z: Prepare system as used by BASIC.
EDIT-Z: Prepare system and load snapshot.

CTRL-F0: Select interrupt mode 0.
CTRL-F1: Select interrupt mode 1.
CTRL-F2: Select interrupt mode 2.
CTRL-F3: Toggle screen ON/OFF (modes 3&4).
CTRL-F4: -
CTRL-F5: -
CTRL-F6: -
CTRL-F7: Execute unconditionally.
CTRL-F8: Stop executing.
CTRL-F9: Execute with current condition.

SHIFT-Cursor_UP: SP = SP - 2
SHIFT-Cursor_DOWN: SP = SP + 2
SHIFT-Cursor_LEFT: -
SHIFT-Cursor_RIGHT: -

DELETE: Delete character in inputs. **EDIT-DELETE**: Delete highlighted instruction.

EDIT-B/C/D/E/H/L/A/F/I/R/X/Y/P/S: Edit B/C/D/E/H/L/A/Flags/I/R/IX/IY/PC/SP

Text editing keys:

A (command mode): Enter new address (command mode only).
D (command mode): Switch to disassembler at cursor's location.
N (command mode): Switch to number mode at cursor's location.
Cursor_Keys: Move cursor around text area.
SHIFT-Cursor_UP: Page up.
SHIFT-Cursor_DOWN: Page down.
RETURN: Toggle command/overtype mode.
Normal keys in overtype mode will poke text at the cursor location.

Number editing keys:

A: Enter new address.
D: Switch to disassembler at cursor's location.
T: Switch to text mode at cursor's location.
Cursor_Keys: Move cursor around text area.
SHIFT-Cursor_UP: Page up.
SHIFT-Cursor_DOWN: Page down.
RETURN: Input number(s) at cursor location / finish inputting numbers.

Disassembler keys (not monitor mode):

A: Input new address.
N: Switch to number mode at cursor's location.
T: Switch to text mode at cursor's location.
SPACE: Save current address.
SHIFT-SPACE: Restore saved address.
Cursor_UP: Up 1 instruction (approx.).
Cursor_DOWN: Down 1 instruction.
SHIFT-Cursor_UP: Page up (approx.).
SHIFT-Cursor_DOWN: Page down.

The Z80 Instruction Set

TurboMON is unique in offering the ENTIRE instruction set without ANY gaps for 'bad opcodes'. Other programs may claim to include undocumented instructions, but they don't normally cover all possibilities. The table at the end of this manual shows all possible combinations of instructions. Instructions in bold text are the normal documented ones.

The TurboMON disassembler knows which instructions can have an index prefix and separates out ones that are not used. They appear in the disassembly as [Unused IX/IY prefix] and have no effect when executed. The Z80 only uses them to signal to use IX/IY instead of HL (in most cases), so LD IX,12345 is really [Signal use IX] LD *HL*,12345. If you have 2 index prefixes following each other, using the previous example you should understand that the second one will over-ride the first one. i.e. [IX prefix] [IY prefix] LD HL,0 will perform LD IY,0 (TurboMON will show the IX prefix as unused, followed by the LD IY,0). Instructions with an ED prefix cannot have an index prefix but ones with a CB prefix can.

102 undocumented instructions that are now widely accepted mainly involve the manipulation of the index registers (blue in the table) as though they were separate 8 bit registers. They were used for a while in Spectrum 'tape loader' security as most people didn't know about them or their disassembler didn't recognise them. The instructions are formed by prefixing instructions involving the H and L registers with an index prefix (DD for IX, FD for IY). The index halves are displayed as IXl, IXh, IYl, IYh. for example: LD A,IXh (formed from LD A,H)

10 of the undocumented instructions are the SLL (shift left logical) instruction that was omitted because it doesn't work properly. Bit 0 of the result is always set after the shift operation.

Other instructions are formed by using an index prefix with CB prefixed instructions that don't use (HL) (hope that was clear!). e.g. Using an IX prefix with SRL B transforms it into: LD B,SRL(IX+d). The instruction performs SRL (IX+d) and loads B with the result. The result is stored back in (IX+d) as well as B, giving 2 instructions for the price of one! Note that the length of the new instructions is always 4 since an index displacement is now required and forms the 3rd byte in the instruction (as with all index instructions).

Because normal BIT instructions only test bits and don't store a result (like SET would), adding a prefix to BIT b,r would transform it into: BIT b,(IX+d) and NOT LD r,BIT b,(IX+d). The fact that the BIT instruction is faster than a SET/RES instruction shows us that the Z80 doesn't try to store a result.

Some instructions come from patterns in the instruction set: NEG, RETN, RETI, IM 0/1/2 are all repeated 8 or 16 bytes after the normal ones.

IN r,(C), with r as (HL), reads a byte from the port in BC but junks the result. It is displayed as IN X,(C). **Note:** The flags are still affected, as with other IN r,(C) instructions.

OUT (C),r, with r as (HL) sends zero to the port in BC. It is displayed as OUT (C),0 to show its effect.

There are spaces for 4 interrupt modes but the Z80 has only 3 (0, 1 and 2). There is a gap between mode 0 and 1 in the instruction set. The missing interrupt mode is another mode 0.

The rest of the instruction set behaves as NOPs, the ones with an ED prefix are obviously 2 byte instructions and will take 8 T-states in reality.

Hex	Dec	Normal	DD/FD Prefix	CB Prefix	ED Prefix	DDCB/FDCB Prefix
00	000	**NOP**	NOP	**RLC B**	NOP	LD B,RLC (IX+d)
01	001	**LD BC,nn**	LD BC,nn	**RLC C**	NOP	LD C,RLC (IX+d)
02	002	**LD (BC),A**	LD (BC),A	**RLC D**	NOP	LD D,RLC (IX+d)
03	003	**INC BC**	INC BC	**RLC E**	NOP	LD E,RLC (IX+d)
04	004	**INC B**	INC B	**RLC H**	NOP	LD H,RLC (IX+d)
05	005	**DEC B**	DEC B	**RLC L**	NOP	LD L,RLC (IX+d)
06	006	**LD B,n**	LD B,n	**RLC (HL)**	NOP	**RLC (IX+d)**
07	007	**RLCA**	RLCA	**RLC A**	NOP	LD A,RLC (IX+d)
08	008	**EX AF,AF'**	EX AF,AF'	**RRC B**	NOP	LD B,RRC (IX+d)
09	009	**ADD HL,BC**	**ADD IX,BC**	**RRC C**	NOP	LD C,RRC (IX+d)
0a	010	**LD A,(BC)**	LD A,(BC)	**RRC D**	NOP	LD D,RRC (IX+d)
0b	011	**DEC BC**	DEC BC	**RRC E**	NOP	LD E,RRC (IX+d)
0c	012	**INC C**	INC C	**RRC H**	NOP	LD H,RRC (IX+d)
0d	013	**DEC C**	DEC C	**RRC L**	NOP	LD L,RRC (IX+d)
0e	014	**LD C,n**	LD C,n	**RRC (HL)**	NOP	**RRC (IX+d)**
0f	015	**RRCA**	RRCA	**RRC A**	NOP	LD A,RRC (IX+d)
10	016	**DJNZ d**	DJNZ d	**RL B**	NOP	LD B,RL (IX+d)
11	017	**LD DE,nn**	LD DE,nn	**RL C**	NOP	LD C,RL (IX+d)
12	018	**LD (DE),A**	LD (DE),A	**RL D**	NOP	LD D,RL (IX+d)
13	019	**INC DE**	INC DE	**RL E**	NOP	LD E,RL (IX+d)
14	020	**INC D**	INC D	**RL H**	NOP	LD H,RL (IX+d)
15	021	**DEC D**	DEC D	**RL L**	NOP	LD L,RL (IX+d)
16	022	**LD D,n**	LD D,n	**RL (HL)**	NOP	**RL (IX+d)**
17	023	**RLA**	RLA	**RL A**	NOP	LD A,RL (IX+d)
18	024	**JR d**	JR d	**RR B**	NOP	LD B,RR (IX+d)
19	025	**ADD HL,DE**	**ADD IX,DE**	**RR C**	NOP	LD C,RR (IX+d)
1a	026	**LD A,(DE)**	LD A,(DE)	**RR D**	NOP	LD D,RR (IX+d)
1b	027	**DEC DE**	DEC DE	**RR E**	NOP	LD E,RR (IX+d)
1c	028	**INC E**	INC E	**RR H**	NOP	LD H,RR (IX+d)
1d	029	**DEC E**	DEC E	**RR L**	NOP	LD L,RR (IX+d)
1e	030	**LD E,n**	LD E,n	**RR (HL)**	NOP	**RR (IX+d)**
1f	031	**RRA**	RRA	**RR A**	NOP	LD A,RR (IX+d)
20	032	**JR NZ,d**	JR NZ,d	**SLA B**	NOP	LD B,SLA (IX+d)
21	033	**LD HL,nn**	**LD IX,nn**	**SLA C**	NOP	LD C,SLA (IX+d)
22	034	**LD (nn),HL**	**LD (nn),IX**	**SLA D**	NOP	LD D,SLA (IX+d)
23	035	**INC HL**	**INC IX**	**SLA E**	NOP	LD E,SLA (IX+d)
24	036	**INC H**	INC IXh	**SLA H**	NOP	LD H,SLA (IX+d)
25	037	**DEC H**	DEC IXh	**SLA L**	NOP	LD L,SLA (IX+d)
26	038	**LD H,n**	LD IXh,n	**SLA (HL)**	NOP	**SLA (IX+d)**
27	039	**DAA**	DAA	**SLA A**	NOP	LD A,SLA (IX+d)
28	040	**JR Z,d**	JR Z,d	**SRA B**	NOP	LD B,SRA (IX+d)
29	041	**ADD HL,HL**	**ADD IX,IX**	**SRA C**	NOP	LD C,SRA (IX+d)
2a	042	**LD HL,(nn)**	**LD IX,(nn)**	**SRA D**	NOP	LD D,SRA (IX+d)
2b	043	**DEC HL**	**DEC IX**	**SRA E**	NOP	LD E,SRA (IX+d)
2c	044	**INC L**	INC IXl	**SRA H**	NOP	LD H,SRA (IX+d)
2d	045	**DEC L**	DEC IXl	**SRA L**	NOP	LD L,SRA (IX+d)
2e	046	**LD L,n**	LD IXl,n	**SRA (HL)**	NOP	**SRA (IX+d)**
2f	047	**CPL**	CPL	**SRA A**	NOP	LD A,SRA (IX+d)
30	048	**JR NC,d**	JR NC,d	SLL B	NOP	LD B,SLL (IX+d)
31	049	**LD SP,nn**	LD SP,nn	SLL C	NOP	LD C,SLL (IX+d)
32	050	**LD (nn),A**	LD (nn),A	SLL D	NOP	LD D,SLL (IX+d)
33	051	**INC SP**	INC SP	SLL E	NOP	LD E,SLL (IX+d)
34	052	**INC (HL)**	**INC (IX+d)**	SLL H	NOP	LD H,SLL (IX+d)
35	053	**DEC (HL)**	**DEC (IX+d)**	SLL L	NOP	LD L,SLL (IX+d)
36	054	**LD (HL),n**	**LD (IX+d),n**	SLL (HL)	NOP	SLL (IX+d)
37	055	**SCF**	SCF	SLL A	NOP	LD A,SLL (IX+d)
38	056	**JR C,d**	JR C,d	**SRL B**	NOP	LD B,SRL (IX+d)
39	057	**ADD HL,SP**	**ADD IX,SP**	**SRL C**	NOP	LD C,SRL (IX+d)
3a	058	**LD A,(nn)**	LD A,(nn)	**SRL D**	NOP	LD D,SRL (IX+d)
3b	059	**DEC SP**	DEC SP	**SRL E**	NOP	LD E,SRL (IX+d)
3c	060	**INC A**	INC A	**SRL H**	NOP	LD H,SRL (IX+d)
3d	061	**DEC A**	DEC A	**SRL L**	NOP	LD L,SRL (IX+d)
3e	062	**LD A,n**	LD A,n	**SRL (HL)**	NOP	**SRL (IX+d)**
3f	063	**CCF**	CCF	**SRL A**	NOP	LD A,SRL (IX+d)

Hex	Dec	Normal	DD/FD Prefix	CB Prefix	ED Prefix	DDCB/FDCB Prefix
40	064	LD B,B	LD B,B	BIT 0,B	IN B,(C)	BIT 0,(IX+d)
41	065	LD B,C	LD B,C	BIT 0,C	OUT (C),B	BIT 0,(IX+d)
42	066	LD B,D	LD B,D	BIT 0,D	SBC HL,BC	BIT 0,(IX+d)
43	067	LD B,E	LD B,E	BIT 0,E	LD (nn),BC	BIT 0,(IX+d)
44	068	LD B,H	LD B,IXh	BIT 0,H	NEG	BIT 0,(IX+d)
45	069	LD B,L	LD B,IXl	BIT 0,L	RETN	BIT 0,(IX+d)
46	070	LD B,(HL)	LD B,(IX+d)	BIT 0,(HL)	IM 0	BIT 0,(IX+d)
47	071	LD B,A	LD B,A	BIT 0,A	LD I,A	BIT 0,(IX+d)
48	072	LD C,B	LD C,B	BIT 1,B	IN C,(C)	BIT 1,(IX+d)
49	073	LD C,C	LD C,C	BIT 1,C	OUT (C),C	BIT 1,(IX+d)
4a	074	LD C,D	LD C,D	BIT 1,D	ADC HL,BC	BIT 1,(IX+d)
4b	075	LD C,E	LD C,E	BIT 1,E	LD BC,(nn)	BIT 1,(IX+d)
4c	076	LD C,H	LD C,IXh	BIT 1,H	NEG	BIT 1,(IX+d)
4d	077	LD C,L	LD C,IXl	BIT 1,L	RETI	BIT 1,(IX+d)
4e	078	LD C,(HL)	LD C,(IX+d)	BIT 1,(HL)	IM 0	BIT 1,(IX+d)
4f	079	LD C,A	LD C,A	BIT 1,A	LD R,A	BIT 1,(IX+d)
50	080	LD D,B	LD D,B	BIT 2,B	IN D,(C)	BIT 2,(IX+d)
51	081	LD D,C	LD D,C	BIT 2,C	OUT (C),D	BIT 2,(IX+d)
52	082	LD D,D	LD D,D	BIT 2,D	SBC HL,DE	BIT 2,(IX+d)
53	083	LD D,E	LD D,E	BIT 2,E	LD (nn),DE	BIT 2,(IX+d)
54	084	LD D,H	LD D,IXh	BIT 2,H	NEG	BIT 2,(IX+d)
55	085	LD D,L	LD D,IXl	BIT 2,L	RETN	BIT 2,(IX+d)
56	086	LD D,(HL)	LD D,(IX+d)	BIT 2,(HL)	IM 1	BIT 2,(IX+d)
57	087	LD D,A	LD D,A	BIT 2,A	LD A,I	BIT 2,(IX+d)
58	088	LD E,B	LD E,B	BIT 3,B	IN E,(C)	BIT 3,(IX+d)
59	089	LD E,C	LD E,C	BIT 3,C	OUT (C),E	BIT 3,(IX+d)
5a	090	LD E,D	LD E,D	BIT 3,D	ADC HL,DE	BIT 3,(IX+d)
5b	091	LD E,E	LD E,E	BIT 3,E	LD DE,(nn)	BIT 3,(IX+d)
5c	092	LD E,H	LD E,IXh	BIT 3,H	NEG	BIT 3,(IX+d)
5d	093	LD E,L	LD E,IXl	BIT 3,L	RETI	BIT 3,(IX+d)
5e	094	LD E,(HL)	LD E,(IX+d)	BIT 3,(HL)	IM 2	BIT 3,(IX+d)
5f	095	LD E,A	LD E,A	BIT 3,A	LD A,R	BIT 3,(IX+d)
60	096	LD H,B	LD IXh,B	BIT 4,B	IN H,(C)	BIT 4,(IX+d)
61	097	LD H,C	LD IXh,C	BIT 4,C	OUT (C),H	BIT 4,(IX+d)
62	098	LD H,D	LD IXh,D	BIT 4,D	SBC HL,HL	BIT 4,(IX+d)
63	099	LD H,E	LD IXh,E	BIT 4,E	LD (nn),HL	BIT 4,(IX+d)
64	100	LD H,H	LD IXh,IXh	BIT 4,H	NEG	BIT 4,(IX+d)
65	101	LD H,L	LD IXh,IXl	BIT 4,L	RETN	BIT 4,(IX+d)
66	102	LD H,(HL)	LD H,(IX+d)	BIT 4,(HL)	IM 0	BIT 4,(IX+d)
67	103	LD H,A	LD IXh,A	BIT 4,A	RRD	BIT 4,(IX+d)
68	104	LD L,B	LD IXl,B	BIT 5,B	IN L,(C)	BIT 5,(IX+d)
69	105	LD L,C	LD IXl,C	BIT 5,C	OUT (C),L	BIT 5,(IX+d)
6a	106	LD L,D	LD IXl,D	BIT 5,D	ADC HL,HL	BIT 5,(IX+d)
6b	107	LD L,E	LD IXl,E	BIT 5,E	LD HL,(nn)	BIT 5,(IX+d)
6c	108	LD L,H	LD IXl,IXh	BIT 5,H	NEG	BIT 5,(IX+d)
6d	109	LD L,L	LD IXl,IXl	BIT 5,L	RETI	BIT 5,(IX+d)
6e	110	LD L,(HL)	LD L,(IX+d)	BIT 5,(HL)	IM 0	BIT 5,(IX+d)
6f	111	LD L,A	LD IXl,A	BIT 5,A	RLD	BIT 5,(IX+d)
70	112	LD (HL),B	LD (IX+d),B	BIT 6,B	IN X,(C)	BIT 6,(IX+d)
71	113	LD (HL),C	LD (IX+d),C	BIT 6,C	OUT (C),0	BIT 6,(IX+d)
72	114	LD (HL),D	LD (IX+d),D	BIT 6,D	SBC HL,SP	BIT 6,(IX+d)
73	115	LD (HL),E	LD (IX+d),E	BIT 6,E	LD (nn),SP	BIT 6,(IX+d)
74	116	LD (HL),H	LD (IX+d),H	BIT 6,H	NEG	BIT 6,(IX+d)
75	117	LD (HL),L	LD (IX+d),L	BIT 6,L	RETN	BIT 6,(IX+d)
76	118	HALT	HALT	BIT 6,(HL)	IM 1	BIT 6,(IX+d)
77	119	LD (HL),A	LD (IX+d),A	BIT 6,A	NOP	BIT 6,(IX+d)
78	120	LD A,B	LD A,B	BIT 7,B	IN A,(C)	BIT 7,(IX+d)
79	121	LD A,C	LD A,C	BIT 7,C	OUT (C),A	BIT 7,(IX+d)
7a	122	LD A,D	LD A,D	BIT 7,D	ADC HL,SP	BIT 7,(IX+d)
7b	123	LD A,E	LD A,E	BIT 7,E	LD SP,(nn)	BIT 7,(IX+d)
7c	124	LD A,H	LD A,IXh	BIT 7,H	NEG	BIT 7,(IX+d)
7d	125	LD A,L	LD A,IXl	BIT 7,L	RETI	BIT 7,(IX+d)
7e	126	LD A,(HL)	LD A,(IX+d)	BIT 7,(HL)	IM 2	BIT 7,(IX+d)
7f	127	LD A,A	LD A,A	BIT 7,A	NOP	BIT 7,(IX+d)

Hex	Dec	Normal	DD/FD Prefix	CB Prefix	ED Prefix	DDCB/FDCB Prefix
80	128	ADD A,B	ADD A,B	RES 0,B	NOP	LD B,RES 0,(IX+d)
81	129	ADD A,C	ADD A,C	RES 0,C	NOP	LD C,RES 0,(IX+d)
82	130	ADD A,D	ADD A,D	RES 0,D	NOP	LD D,RES 0,(IX+d)
83	131	ADD A,E	ADD A,E	RES 0,E	NOP	LD E,RES 0,(IX+d)
84	132	ADD A,H	ADD A,IXh	RES 0,H	NOP	LD H,RES 0,(IX+d)
85	133	ADD A,L	ADD A,IXl	RES 0,L	NOP	LD L,RES 0,(IX+d)
86	134	ADD A,(HL)	ADD A,(IX+d)	RES 0,(HL)	NOP	RES 0,(IX+d)
87	135	ADD A,A	ADD A,A	RES 0,A	NOP	LD A,RES 0,(IX+d)
88	136	ADC A,B	ADC A,B	RES 1,B	NOP	LD B,RES 1,(IX+d)
89	137	ADC A,C	ADC A,C	RES 1,C	NOP	LD C,RES 1,(IX+d)
8a	138	ADC A,D	ADC A,D	RES 1,D	NOP	LD D,RES 1,(IX+d)
8b	139	ADC A,E	ADC A,E	RES 1,E	NOP	LD E,RES 1,(IX+d)
8c	140	ADC A,H	ADC A,IXh	RES 1,H	NOP	LD H,RES 1,(IX+d)
8d	141	ADC A,L	ADC A,IXl	RES 1,L	NOP	LD L,RES 1,(IX+d)
8e	142	ADC A,(HL)	ADC A,(IX+d)	RES 1,(HL)	NOP	RES 1,(IX+d)
8f	143	ADC A,A	ADC A,A	RES 1,A	NOP	LD A,RES 1,(IX+d)
90	144	SUB B	SUB B	RES 2,B	NOP	LD B,RES 2,(IX+d)
91	145	SUB C	SUB C	RES 2,C	NOP	LD C,RES 2,(IX+d)
92	146	SUB D	SUB D	RES 2,D	NOP	LD D,RES 2,(IX+d)
93	147	SUB E	SUB E	RES 2,E	NOP	LD E,RES 2,(IX+d)
94	148	SUB H	SUB IXh	RES 2,H	NOP	LD H,RES 2,(IX+d)
95	149	SUB L	SUB IXl	RES 2,L	NOP	LD L,RES 2,(IX+d)
96	150	SUB (HL)	SUB (IX+d)	RES 2,(HL)	NOP	RES 2,(IX+d)
97	151	SUB A	SUB A	RES 2,A	NOP	LD A,RES 2,(IX+d)
98	152	SBC A,B	SBC A,B	RES 3,B	NOP	LD B,RES 3,(IX+d)
99	153	SBC A,C	SBC A,C	RES 3,C	NOP	LD C,RES 3,(IX+d)
9a	154	SBC A,D	SBC A,D	RES 3,D	NOP	LD D,RES 3,(IX+d)
9b	155	SBC A,E	SBC A,E	RES 3,E	NOP	LD E,RES 3,(IX+d)
9c	156	SBC A,H	SBC A,IXh	RES 3,H	NOP	LD H,RES 3,(IX+d)
9d	157	SBC A,L	SBC A,IXl	RES 3,L	NOP	LD L,RES 3,(IX+d)
9e	158	SBC A,(HL)	SBC A,(IX+d)	RES 3,(HL)	NOP	RES 3,(IX+d)
9f	159	SBC A,A	SBC A,A	RES 3,A	NOP	LD A,RES 3,(IX+d)
a0	160	AND B	AND B	RES 4,B	LDI	LD B,RES 4,(IX+d)
a1	161	AND C	AND C	RES 4,C	CPI	LD C,RES 4,(IX+d)
a2	162	AND D	AND D	RES 4,D	INI	LD D,RES 4,(IX+d)
a3	163	AND E	AND E	RES 4,E	OUTI	LD E,RES 4,(IX+d)
a4	164	AND H	AND IXh	RES 4,H	NOP	LD H,RES 4,(IX+d)
a5	165	AND L	AND IXl	RES 4,L	NOP	LD L,RES 4,(IX+d)
a6	166	AND (HL)	AND (IX+d)	RES 4,(HL)	NOP	RES 4,(IX+d)
a7	167	AND A	AND A	RES 4,A	NOP	LD A,RES 4,(IX+d)
a8	168	XOR B	XOR B	RES 5,B	LDD	LD B,RES 5,(IX+d)
a9	169	XOR C	XOR C	RES 5,C	CPD	LD C,RES 5,(IX+d)
aa	170	XOR D	XOR D	RES 5,D	IND	LD D,RES 5,(IX+d)
ab	171	XOR E	XOR E	RES 5,E	OUTD	LD E,RES 5,(IX+d)
ac	172	XOR H	XOR IXh	RES 5,H	NOP	LD H,RES 5,(IX+d)
ad	173	XOR L	XOR IXl	RES 5,L	NOP	LD L,RES 5,(IX+d)
ae	174	XOR (HL)	XOR (IX+d)	RES 5,(HL)	NOP	RES 5,(IX+d)
af	175	XOR A	XOR A	RES 5,A	NOP	LD A,RES 5,(IX+d)
b0	176	OR B	OR B	RES 6,B	LDIR	LD B,RES 6,(IX+d)
b1	177	OR C	OR C	RES 6,C	CPIR	LD C,RES 6,(IX+d)
b2	178	OR D	OR D	RES 6,D	INIR	LD D,RES 6,(IX+d)
b3	179	OR E	OR E	RES 6,E	OTIR	LD E,RES 6,(IX+d)
b4	180	OR H	OR IXh	RES 6,H	NOP	LD H,RES 6,(IX+d)
b5	181	OR L	OR IXl	RES 6,L	NOP	LD L,RES 6,(IX+d)
b6	182	OR (HL)	OR (IX+d)	RES 6,(HL)	NOP	RES 6,(IX+d)
b7	183	OR A	OR A	RES 6,A	NOP	LD A,RES 6,(IX+d)
b8	184	CP B	CP B	RES 7,B	LDDR	LD B,RES 7,(IX+d)
b9	185	CP C	CP C	RES 7,C	CPDR	LD C,RES 7,(IX+d)
ba	186	CP D	CP D	RES 7,D	INDR	LD D,RES 7,(IX+d)
bb	187	CP E	CP E	RES 7,E	OTDR	LD E,RES 7,(IX+d)
bc	188	CP H	CP IXh	RES 7,H	NOP	LD H,RES 7,(IX+d)
bd	189	CP L	CP IXl	RES 7,L	NOP	LD L,RES 7,(IX+d)
be	190	CP (HL)	CP (IX+d)	RES 7,(HL)	NOP	RES 7,(IX+d)
bf	191	CP A	CP A	RES 7,A	NOP	LD A,RES 7,(IX+d)

Hex	Dec	Normal	DD/FD Prefix	CB Prefix	ED Prefix	DDCB/FDCB Prefix
c0	192	**RET NZ**	RET NZ	**SET 0,B**	NOP	LD B,SET 0,(IX+d)
c1	193	**POP BC**	POP BC	**SET 0,C**	NOP	LD C,SET 0,(IX+d)
c2	194	**JP NZ,nn**	JP NZ,nn	**SET 0,D**	NOP	LD D,SET 0,(IX+d)
c3	195	**JP nn**	JP nn	**SET 0,E**	NOP	LD E,SET 0,(IX+d)
c4	196	**CALL NZ,nn**	CALL NZ,nn	**SET 0,H**	NOP	LD H,SET 0,(IX+d)
c5	197	**PUSH BC**	PUSH BC	**SET 0,L**	NOP	LD L,SET 0,(IX+d)
c6	198	**ADD A,n**	ADD A,n	**SET 0,(HL)**	NOP	**SET 0,(IX+d)**
c7	199	**RST 0**	RST 0	**SET 0,A**	NOP	LD A,SET 0,(IX+d)
c8	100	**RET Z**	RET Z	**SET 1,B**	NOP	LD B,SET 1,(IX+d)
c9	201	**RET**	RET	**SET 1,C**	NOP	LD C,SET 1,(IX+d)
ca	202	**JP Z,nn**	JP Z,nn	**SET 1,D**	NOP	LD D,SET 1,(IX+d)
cb	203	**[See CB Prefix]**	[See DDCB/FDCB]	**SET 1,E**	NOP	LD E,SET 1,(IX+d)
cc	204	**CALL Z,nn**	CALL Z,nn	**SET 1,H**	NOP	LD H,SET 1,(IX+d)
cd	205	**CALL nn**	CALL nn	**SET 1,L**	NOP	LD L,SET 1,(IX+d)
ce	206	**ADC A,n**	ADC A,n	**SET 1,(HL)**	NOP	**SET 1,(IX+d)**
cf	207	**RST 8**	RST 8	**SET 1,A**	NOP	LD A,SET 1,(IX+d)
d0	208	**RET NC**	RET NC	**SET 2,B**	NOP	LD B,SET 2,(IX+d)
d1	209	**POP DE**	POP DE	**SET 2,C**	NOP	LD C,SET 2,(IX+d)
d2	210	**JP NC,nn**	JP NC,nn	**SET 2,D**	NOP	LD D,SET 2,(IX+d)
d3	211	**OUT (n),A**	OUT (n),A	**SET 2,E**	NOP	LD E,SET 2,(IX+d)
d4	212	**CALL NC,nn**	CALL NC,nn	**SET 2,H**	NOP	LD H,SET 2,(IX+d)
d5	213	**PUSH DE**	PUSH DE	**SET 2,L**	NOP	LD L,SET 2,(IX+d)
d6	214	**SUB n**	SUB n	**SET 2,(HL)**	NOP	**SET 2,(IX+d)**
d7	215	**RST 10H**	RST 10H	**SET 2,A**	NOP	LD A,SET 2,(IX+d)
d8	216	**RET C**	RET C	**SET 3,B**	NOP	LD B,SET 3,(IX+d)
d9	217	**EXX**	EXX	**SET 3,C**	NOP	LD C,SET 3,(IX+d)
da	218	**JP C,nn**	JP C,nn	**SET 3,D**	NOP	LD D,SET 3,(IX+d)
db	219	**IN A,(n)**	IN A,(n)	**SET 3,E**	NOP	LD E,SET 3,(IX+d)
dc	220	**CALL C,nn**	CALL C,nn	**SET 3,H**	NOP	LD H,SET 3,(IX+d)
dd	221	**[IX Prefix]**	[IX Prefix]	**SET 3,L**	NOP	LD L,SET 3,(IX+d)
de	222	**SBC A,n**	SBC A,n	**SET 3,(HL)**	NOP	**SET 3,(IX+d)**
df	223	**RST 18H**	RST 18H	**SET 3,A**	NOP	LD A,SET 3,(IX+d)
e0	224	**RET PO**	RET PO	**SET 4,B**	NOP	LD B,SET 4,(IX+d)
e1	225	**POP HL**	**POP IX**	**SET 4,C**	NOP	LD C,SET 4,(IX+d)
e2	226	**JP PO,nn**	JP PO,nn	**SET 4,D**	NOP	LD D,SET 4,(IX+d)
e3	227	**EX (SP),HL**	**EX (SP),IX**	**SET 4,E**	NOP	LD E,SET 4,(IX+d)
e4	228	**CALL PO,nn**	CALL PO,nn	**SET 4,H**	NOP	LD H,SET 4,(IX+d)
e5	229	**PUSH HL**	**PUSH IX**	**SET 4,L**	NOP	LD L,SET 4,(IX+d)
e6	230	**AND n**	AND n	**SET 4,(HL)**	NOP	**SET 4,(IX+d)**
e7	231	**RST 20H**	RST 20H	**SET 4,A**	NOP	LD A,SET 4,(IX+d)
e8	232	**RET PE**	RET PE	**SET 5,B**	NOP	LD B,SET 5,(IX+d)
e9	233	**JP (HL)**	**JP (IX)**	**SET 5,C**	NOP	LD C,SET 5,(IX+d)
ea	234	**JP PE,nn**	JP PE,nn	**SET 5,D**	NOP	LD D,SET 5,(IX+d)
eb	235	**EX DE,HL**	EX DE,HL	**SET 5,E**	NOP	LD E,SET 5,(IX+d)
ec	236	**CALL PE,nn**	CALL PE,nn	**SET 5,H**	NOP	LD H,SET 5,(IX+d)
ed	237	**[See ED Prefix]**	[Ignore DD/FD]	**SET 5,L**	NOP	LD L,SET 5,(IX+d)
ee	238	**XOR n**	XOR n	**SET 5,(HL)**	NOP	**SET 5,(IX+d)**
ef	239	**RST 28H**	RST 28H	**SET 5,A**	NOP	LD A,SET 5,(IX+d)
f0	240	**RET P**	RET P	**SET 6,B**	NOP	LD B,SET 6,(IX+d)
f1	241	**POP AF**	POP AF	**SET 6,C**	NOP	LD C,SET 6,(IX+d)
f2	242	**JP P,nn**	JP P,nn	**SET 6,D**	NOP	LD D,SET 6,(IX+d)
f3	243	**DI**	DI	**SET 6,E**	NOP	LD E,SET 6,(IX+d)
f4	244	**CALL P,nn**	CALL P,nn	**SET 6,H**	NOP	LD H,SET 6,(IX+d)
f5	245	**PUSH AF**	PUSH AF	**SET 6,L**	NOP	LD L,SET 6,(IX+d)
f6	246	**OR n**	OR n	**SET 6,(HL)**	NOP	**SET 6,(IX+d)**
f7	247	**RST 30H**	RST 30H	**SET 6,A**	NOP	LD A,SET 6,(IX+d)
f8	248	**RET M**	RET M	**SET 7,B**	NOP	LD B,SET 7,(IX+d)
f9	249	**LD SP,HL**	**LD SP,IX**	**SET 7,C**	NOP	LD C,SET 7,(IX+d)
fa	250	**JP M,nn**	JP M,nn	**SET 7,D**	NOP	LD D,SET 7,(IX+d)
fb	251	**EI**	EI	**SET 7,E**	NOP	LD E,SET 7,(IX+d)
fc	252	**CALL M,nn**	CALL M,nn	**SET 7,H**	NOP	LD H,SET 7,(IX+d)
fd	253	**[IY Prefix]**	[IY Prefix]	**SET 7,L**	NOP	LD L,SET 7,(IX+d)
fe	254	**CP n**	CP n	**SET 7,(HL)**	NOP	**SET 7,(IX+d)**
ff	255	**RST 38H**	RST 38H	**SET 7,A**	NOP	LD A,SET 7,(IX+d)