



TSK80x MCU

Summary

Core Reference
CR0117 (v1.6) August 28, 2006

The TSK80x is a fully functional, 8-bit microcontroller, incorporating the von Neumann architecture. This core reference includes architectural and hardware descriptions, instruction sets and on-chip debugging functionality for the TSK80x family.

The TSK80x is a fully functional 8-bit embedded processor which is instruction set compatible with the Zilog Z80CPU¹. The TSK80x supports hardware interrupts, halt and wait states for low speed memory and I/O devices.

Important Notice: Supply of this soft core under the terms and conditions of the Altium End-User License Agreement does not convey nor imply any patent rights to the supplied technologies. Users are cautioned that a license may be required for any use covered by such patent rights.

Features

- Control Unit
 - 8-bit Instruction decoder
- Arithmetic Logic Unit
 - 8-bit arithmetic and logical operations
 - 16-bit arithmetic operations
 - Boolean manipulations
- Register File Unit
 - Duplicate set of both general purpose and flag registers
 - Two 16-bit index registers
- Interrupt Controller
 - Three modes of maskable interrupts
 - Non-maskable interrupt
- External Memory Interface
 - Can address up to 64KB of Program memory
 - Can address up to 64KB of Data memory

¹ The TSK80A is instruction set compatible. The TSK80A_D is instruction set compatible with the exception of instruction LD H, H. This opcode is reserved and is used to represent a software breakpoint.

TSK80x MCU

- Can address up to 64KB of external I/O peripheral devices

Available Devices

Both standard and debug-enabled (OCD) versions of the microcontroller are available – the TSK80A and TSK80A_D respectively.

Note: Unless specified otherwise, the feature/description described for the standard version of the controller applies to the debug-enabled version in exactly the same way.

These devices can be found in the FPGA Processors integrated library (\Program Files\Altium Designer 6\Library\Fpga\FPGA Processors.IntLib).

Architectural overview

Symbols

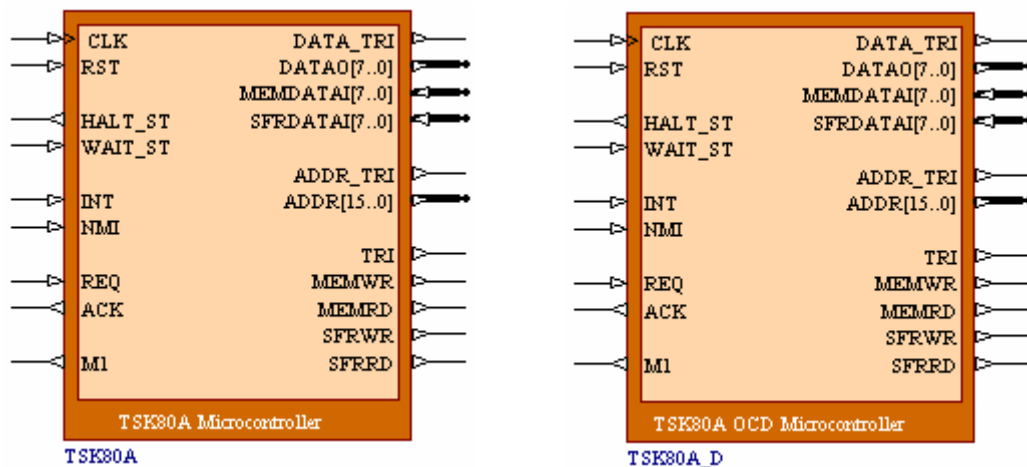


Figure 1. TSK80x family symbols

Pin description

The pinout of the TSK80x is described in Table 1 below.

Table 1. TSK80x pin description

Name	Type	Polarity/Bus size	Description
CPU Control Signals			
CLK	I	Rise	External system clock (used for internal clock counters and all other synchronous circuitry).
RST	I	High	External system reset. A High on this pin for at least one clock cycle while the external system clock (CLK) is running resets the device.
HALT_ST	O	High	A High state on this pin indicates that the CPU has executed a Halt instruction and is awaiting either a non-maskable or maskable interrupt before operation can resume.
WAIT_ST	I	High	A High on this pin indicates to the CPU that the addressed memory or I/O device is not ready for a data transfer. The CPU continues to enter a wait state as long as this signal is active.

Name	Type	Polarity/Bus size	Description
INT	I	High	Maskable Interrupt Request. This signal is generated by the I/O device. The CPU honors a request at the end of the current instruction, if the internal software-controlled interrupt enable flip-flop is enabled.
NMI	I	Rise	Non-maskable Interrupt Request. This signal has a higher priority than INT and is always recognized at the end of the current instruction, irrespective of the status of the interrupt enable flip-flop, and forces the CPU to restart at address 0066h.
System Control Signals			
M1	O	High	This signal is used to distinguish between a memory fetch operation and a normal data memory read operation. When active together with MEMRD, it indicates that the current machine cycle is the Opcode fetch cycle of an instruction execution
TRI	O	Low	Tri-state bus control. This signal is used to enable off-core tri-state buffering for the MEMWR, MEMRD, SFRWR and SFRRD signals.
MEMWR	O	High	This signal indicates that the CPU data bus holds valid data to be stored at the addressed memory
MEMRD	O	High	This signal indicates that the CPU wants to read data from memory and memory should use this signal to gate data onto the CPU data bus
SFRWR	O	High	This signal indicates that the CPU data bus holds valid data to be written to the I/O device
SFRRD	O	High	This signal indicates that the CPU wants to read data from an I/O device and the device should use this signal to gate data onto the CPU data bus
Bus Control Signals			
REQ	I	High	This signal has a higher priority than NMI and is always recognized at the end of the current machine cycle. When this signal is active, the CPU address bus, data bus and associated control signals are forced to go to a high-impedance state, so that other devices can take control of these lines.

Name	Type	Polarity/Bus size	Description
ACK	O	High	When this signal is active, it indicates to the requesting device that the CPU address bus, data bus and associated control signals have entered their high-impedance state and it can now take control of these lines.
Data Bus Signals			
DATA_TRI	O	Low	Data bus tristate control
DATAO	O	8	Data bus output
MEMDATAI	I	8	Data bus input from external memory space
SFRDATAI	I	8	Data bus input from external peripheral I/O space
Address Bus Signals			
ADDR_TRI	O	Low	Address bus tristate control
ADDR	O	16	Address bus output. This 16-bit signal specifies I/O and memory addresses to be accessed. The bus is an input when the external master is accessing the on-chip peripherals

Memory organization

The TSK80x has a 16-bit address bus capable of addressing up to 64KB of memory space. This same memory space is used to store both code (program) and data. The microcontroller does not distinguish between Program and Data memory space; if a different space is required for Data and Program memory, separate external memory blocks must be used. When separate memory blocks are used, an additional external device must be used in order to switch between the two.

The TSK80A can also communicate with external peripheral devices. To address I/O ports, it uses the same 16-bit address bus, so the addressable I/O space is also 64KB. However, full access in both directions is only available within the first 256 bytes (0000h – 00FFh).

The memory and I/O space are different. This is described in more detail in the following sections.

Memory map

The TSK80x can address up to 64KB of memory space, from 0000h to FFFFh. After a reset, the CPU starts program execution from location 0000h. From this location, the first instruction Opcode is fetched and executed.

The lower part of the Program memory includes a non-maskable interrupt (stored at address 0066h), a maskable interrupt vector in mode 1 (stored at address 0038h) and all restart vectors.

When the CPU reads data from memory, the MEMRD signal goes to an active state (High) and then memory is read. When an instruction Opcode is read, an additional signal, M1, goes into active state

TSK80x MCU

(High). Controlling the state of M1 gives information on what kind of memory access is being processed.

The memory map is illustrated in the figure below.

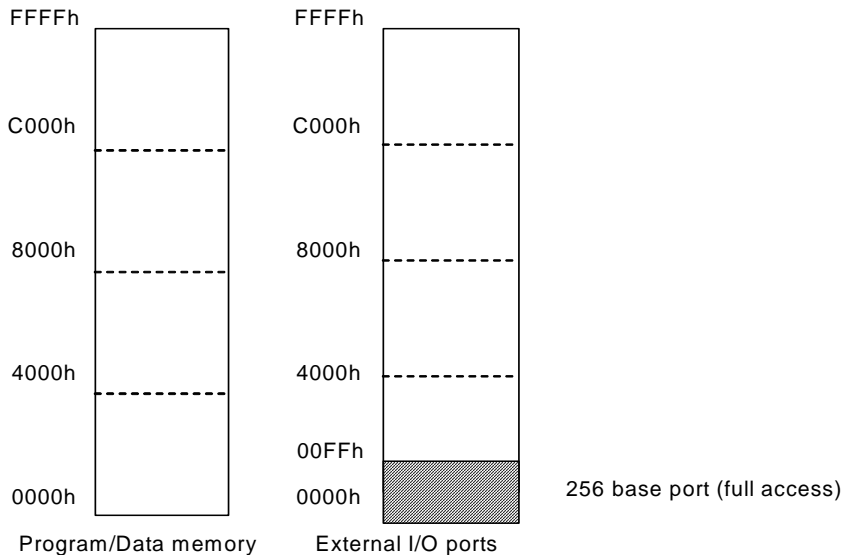


Figure 2. Memory map

Memory fetch operation

Signal M1 is used to distinguish between normal data read from memory and an instruction fetch. It goes to an active state (High) on the rising edge of the clock in cycle 1 when the instruction fetch cycle starts and it returns to invalid state on the rising edge of the clock in cycle 2, when the fetch cycle finishes.

In summary, a memory fetch operation is distinguished from a memory read by the active state of signal M1.

For slow memory devices, the CPU can add additional cycles and wait on data.

External I/O ports space

The TSK80x can address up to 64KB of external I/O space, from 0000h to FFFFh. Only ports addressed in the lowest 256 bytes of this space are fully accessible (both read and write). Addressing of ports above this range is made possible by special features inherent to the execution of the input and output instructions.

When addressing I/O space, all signals on the address bus have an active state, but only the lowest eight are valid (equating to 256 bytes). One of the general purpose registers is passed on the top half of the address bus at the same time. This behavior makes it possible to utilize all 64KB of I/O space.

During instructions IN A,(n) and OUT (n),A, immediate data n is passed on to the low order byte of the address bus and the contents of the Accumulator are passed on to the high order byte. This feature gives free access to any port in the entire 64KB I/O address space, but only when reading data. When

writing data, the data that is to be written (contained in the Accumulator and subsequently put onto the data bus) is the same value that is put onto the high order byte of the address bus. The range of addressable space is therefore limited by the value of the data that is to be written. For example, if the data to be written is FFh (stored in the Accumulator), you could conceivably use any value in the range 00h – FFh for n and hence write to any port in the I/O space (0000h – FFFFh). However, if the data to be written from the Accumulator was 20h, then the addressable I/O space would be limited to the range 0000h – 20FFh.

The instructions IN r,(C) and OUT (C), r are more flexible, but they have limitations too. In these instructions, the contents of register C are placed on to the low order byte of the address bus and the contents of register B on to the high order byte.

Hardware description

Block diagram

Figure 3 shows the block diagram for the TSK80x. The processor is essentially divided into four sub-blocks:

- the main State Machine (FSM)
- the Decoder
- the ALU
- the Internal Registers.

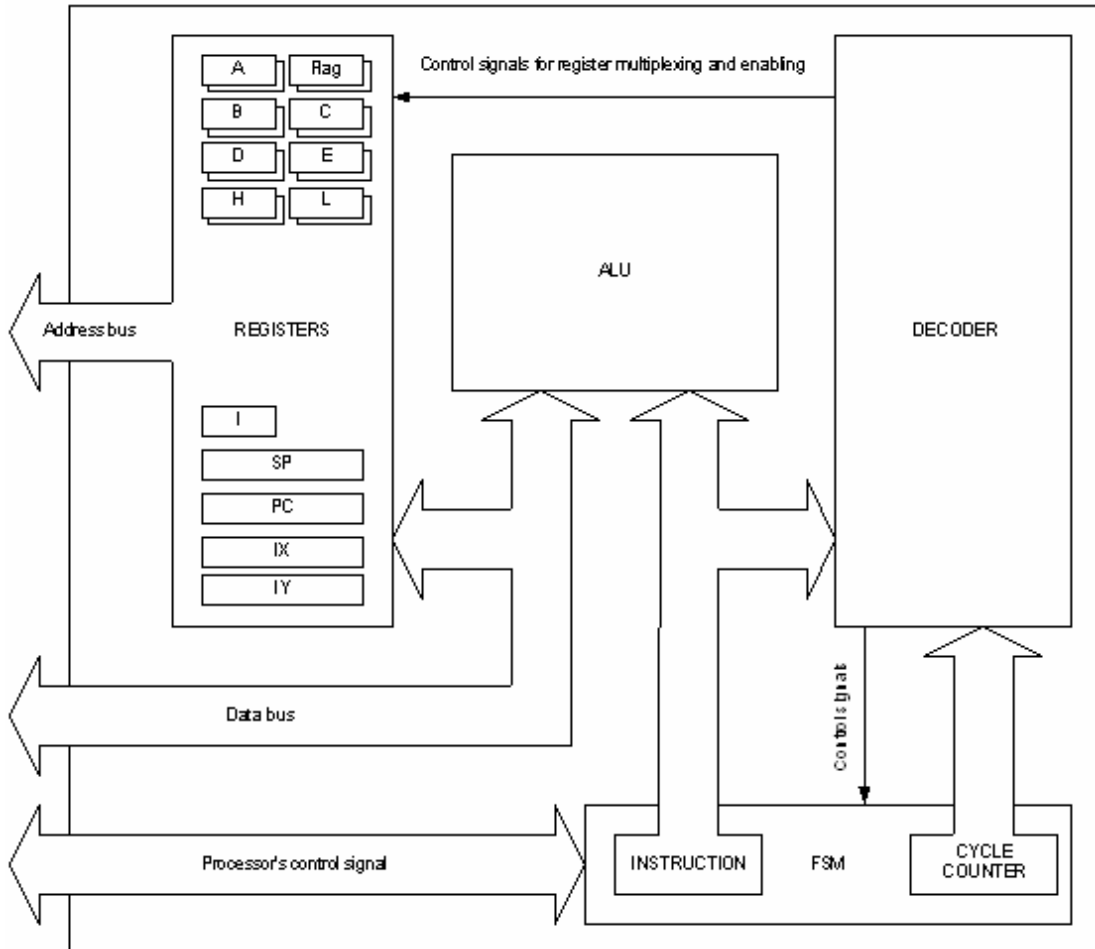


Figure 3. TSK80x block diagram

Finite State Machine (FSM)

The processor's state machine has two main functions – it controls the processor's control outputs during an instruction cycle and also manages exception vectors (interrupts, halt and external bus request).

The state machine controls all access to external memory and peripheral devices. It also ensures that the current instruction has completed before an exception is processed and executed.

Figure 4 summarizes the states involved in an instruction cycle and the transitions between each.

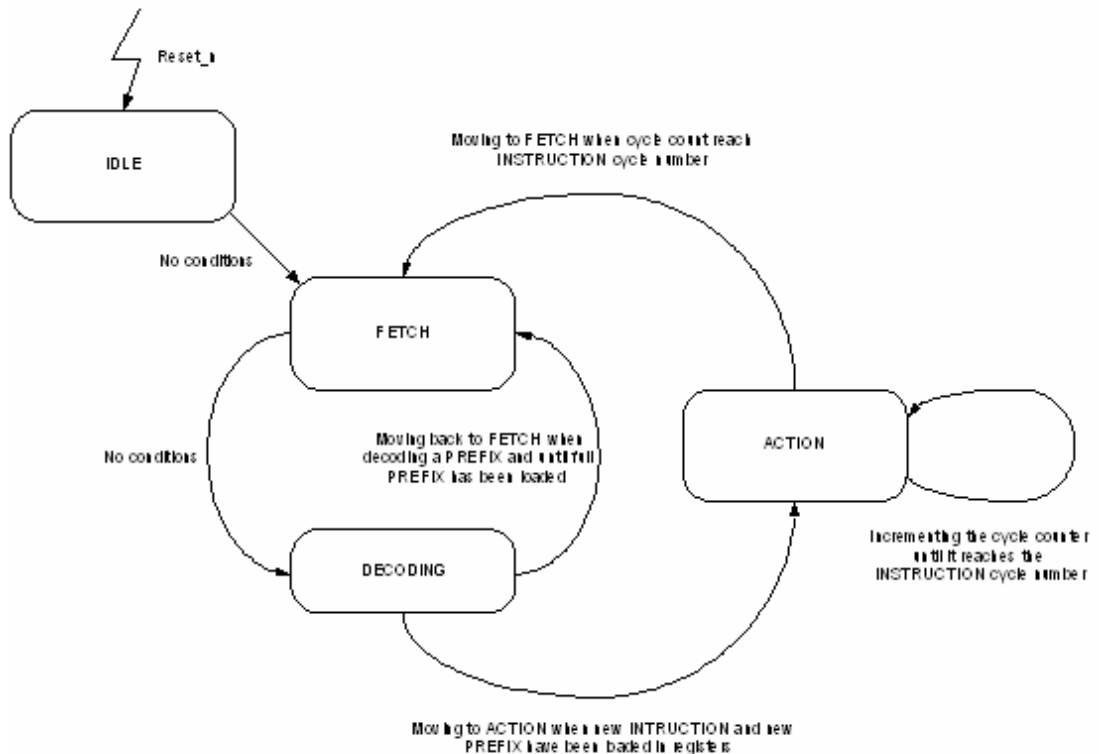


Figure 4. TSK80x main state machine: Instruction cycle

The **IDLE** state is the initial state, entered upon reception of an external reset (**RST**). As can be seen, the actual instruction cycle consists of the following three states (stages):

- **FETCH** – the processor loads a new instruction byte from memory
- **DECODING** – the processor interprets the instruction byte loaded during the **FETCH** stage, determines if it is a prefix byte and, if it is, returns to the **FETCH** state to load the rest of the instruction. After the full instruction has been loaded, the processor enters the **ACTION** state, to start the execution of the instruction
- **ACTION** – the processor activates the relevant control signals to complete the instruction. In the case of a multi-clock cycle instruction (e.g. an addition between two registers) the state machine increments the cycle counter, activating the relevant control signals as required.

Decoder

Where the state machine controls external access and general data flow inside the processor, the Decoder controls all of the instruction-specific internal control signals. The state machine passes the value of a newly-loaded instruction to the Decoder. The Decoder uses this value to generate the required internal control signals.

Upon entering the ACTION state, the Decoder sends the number of cycles required for the instruction, back to the state machine. The state machine increments its internal cycle counter until it reaches this value, after which it knows that the current instruction has completed and it can FETCH the next instruction from memory. Figure 5 summarizes the interaction between the state machine and the Decoder.

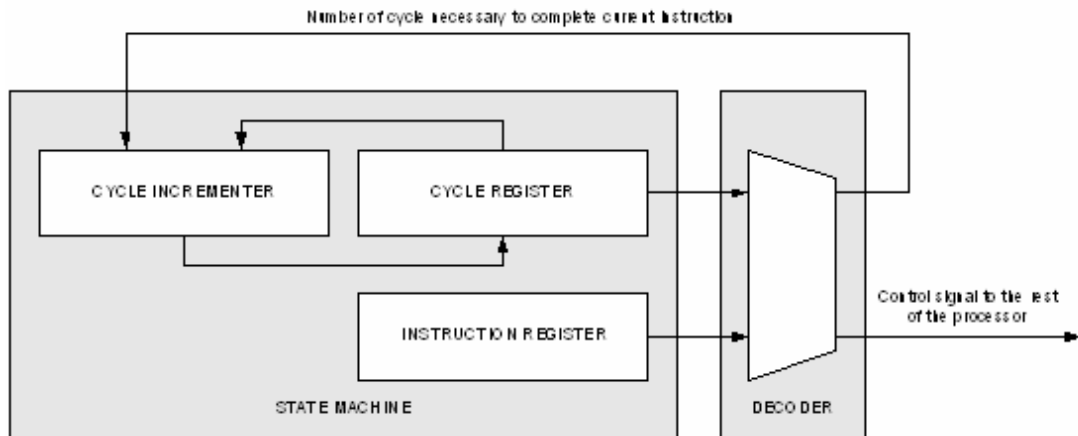


Figure 5. FSM-Decoder interaction in the TSK80x

ALU

The Arithmetic Logic Unit (ALU) is involved in every operation on the data. The CPU executes the following operations:

- Add 8-bit data with or without Carry flag
- Subtract 8-bit data with or without Carry flag
- Increment and decrement 8-bit data
- Logic operation AND, OR, XOR on 8-bit data
- Compare 8-bit data with Accumulator
- Shift, rotate 8-bit data with or without Carry flag
- Set, reset or test any bit in 8-bit data
- Convert Accumulator into packed BCD
- Negate Accumulator ($0 - A$)
- Logic operation NOT on data in Accumulator
- Add 16-bit data with or without Carry flag
- Subtract 16-bit data with Carry flag

- Increment and decrement 16-bit data.

The result of all 8-bit operations is stored in the Accumulator and reflected in the status of various flags in the flag register (F).

For 16-bit operations, the result is stored in register HL (which acts as the Accumulator). For instructions with an Opcode prefix of DD or FD, the result is stored in the IX and IY registers respectively.

The Flag register (F) tests the following conditions:

- S - sign (first bit of result)
- Z - zero (is set (1) if the result of the operation is 0)
- H – half carry (is set (1) if the add or subtract operation produced a carry into or a borrow from bit 4 of the result)
- P/V – parity/overflow (is set (1) if the result of the operation is even or produced an overflow)
- N – add/subtract (is set (1) if last executed operation was a subtract)
- C – carry (set (1) if the operation produced a carry from the MSB of the result).

The Flag register can also be changed by other instructions not connected with logical or arithmetic operations. In such cases, the conditions tested as listed above may not necessarily apply. Examples are block transfer instructions or set carry flag instructions (SCF) that only write to the Carry flag.

Registers

All registers in the TSK80x can be changed under program control. They can be split into three groups, as described below.

The first group consists of two duplicate sets of 8-bit registers - base and alternative. Only one set can be used at a time. Switching between sets is executed by software (instructions EXX and EX AF, AF'). Both sets in this group consist of the following registers:

- A (Accumulator)
- F (Flag register)
- B, C, D, E, H and L (General Purpose registers)

Although they are 8-bit registers, they can be treated as 16-bit registers. In this case, registers can be joined in the following pairings: B and C as BC; D and E as DE; H and L as HL; A and F as AF.

The second group consists of five registers with assigned functions. These are:

- Index registers (IX and IY) - used in index addressing mode
- Stack Pointer (SP) – used to store address on top of Stack
- Program Counter (PC) – used to store address of next instruction to be executed
- Interrupt register (I) – used to store the interrupt address and the device requesting the service

The third group consists of two Interrupt Status registers (IFF1, IFF2) and the Interrupt Mode register (IM). The Interrupt Status registers store information as to whether a maskable interrupt request should be serviced or not. Its state can be changed by the Enable Interrupt instruction (EI) and Disable Interrupt instruction (DI). The actual state can be tested only indirectly with the results of executed instruction LD A,I which changes the flag P/V depending on the state of the register IFF2. Additionally,

when servicing non-maskable interrupts, the state of register IFF1 is stored into IFF2 and IFF1 is reset (disable maskable interrupts).

The RETN instruction is used for copying the contents of IFF2 back into IFF1 (setting IFF1 back to its original state, prior to the non-maskable interrupt being serviced).

The IM register is less flexible (no way to test its value). Its state can be set only by execution of instructions IM0, IM1, or IM2. Each one sets a different interrupt service mode.

The B, C, D and E registers

The B, C, D and E registers are 8-bit registers set to 00h after a reset of the processor. They are used as general purpose registers to store data. In some of the instructions, they are connected in a pair (B to C and D to E) and are then treated as 16-bit registers. As a 16-bit register, they can contain a 16-bit data value, be used as an address register (storing an address of a location in memory space), or act as a transfer counter (BC pair only).

The H and L registers

The H and L registers are 8-bit registers set to 00h after a reset of the processor. They are used as general purpose registers to store data and as a 16-bit register pair to store the address of an operand. As an address register, HL is more flexible than register pairs BC and DE. Almost all operations appear in addressing mode using register HL. In transfer data operations, the HL register is used as the source or destination address register. For 16-bit arithmetic operations, HL is used as a register that stores the result of an operation (acting as the Accumulator).

The IX and IY registers

The Index registers (IX and IY) are 16-bit registers set to 0000h after a reset of the processor. They mainly store addresses used in indexed addressing mode but can also be used as general purpose registers. They can be used the same way as register HL but not in instructions which operate on 8-bit data. (There are no instructions that operate on the high or low order bytes of registers IX and IY only).

Program Counter (PC)

The Program Counter (PC) is a 16-bit wide register that is set to 0000h after a reset of the processor. Its value is incremented in every FETCH cycle after reading the instruction Opcode.

Stack Pointer (SP)

The Stack Pointer (SP) is a 16-bit register set to 0000h after a reset of the processor. It is automatically incremented at the beginning of execution of instructions POP or RET and is decremented in instructions PUSH, CALL and RST. The SP always contains the address of the last stored data on the external memory stack.

Interrupt register (I)

The Interrupt register (I) is an 8-bit register set to 00h after a reset of the processor. It stores the basic addresses of the interrupts vector table.

In an interrupt acknowledge cycle, in interrupt mode 2, its value (placed in high order byte of address bus) along with data from the serviced device (placed in low order byte of address bus) comprises the address of the first instruction of the interrupt service routine.

Bus request cycle

On the rising edge of the last clock cycle of every machine cycle, the state of signal REQ is tested. If it is Low, no special action is made. If it is High, the CPU disconnects itself from all buses.

Signal ACK is set to High to show other devices that the processor is disconnected from the system and other devices can take control of all lines. The rest of the signals are set to a non-active state. The exception is signal HALT_ST which does not change state (if it was High it will remain High, if it was Low, it will stay Low).

The CPU remains in this state, only testing the state of signal REQ (on every rising edge of the clock). When the tested signal changes to Low, the CPU leaves this suspended state and resumes program execution from the next instruction. If the CPU was in the middle of executing an instruction before it was disconnected from the bus lines, it will not resume execution of this same instruction, rather it will resume by executing the subsequent instruction. On the falling edge of the clock, signal ACK (at the same time finding a Low on signal REQ) is set to Low.

The REQ signal has higher priority than interrupts and, during a bus request/acknowledge cycle, the processor cannot be interrupted.

Note: Off-core tristate buffers can be wired into a design in order to place bus signals in a high impedance state. Tristate control signals are provided for this very purpose:

- ADDR_TRI for the ADDR bus
- DATA_TRI for the DATAO, MEMDATAI and SFRDATAI buses
- TRI for the MEMWR, MEMRD, SFRWR and SFRRD lines.

Halt Acknowledge

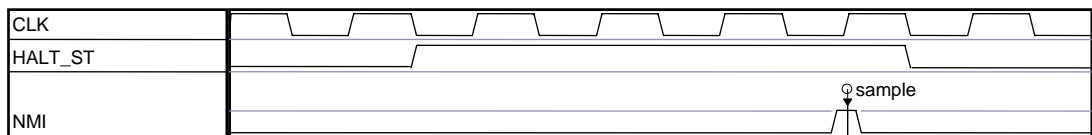


Figure 6. Halt acknowledge

Note: sample - point at which signal is tested.

The result of the execution of the HALT instruction is that the CPU goes into a loop, which looks as though the NOP instruction is continuously being executed. The difference is that the CPU does not fetch the next instruction. Instead, the Program Counter does not change its value and the HALT_ST signal is set to High (on the falling edge of the fourth clock cycle in the machine cycle in which the HALT instruction is executed).

The processor does not take any action. It tests interrupt signals on the rising edge of the fourth clock cycle in each machine cycle. The processor can be in this state without any time restriction, but if the CPU starts to service an interrupt, this cycle is suspended and the CPU returns to normal work.

Hardware Reset (RST)

All registers are synchronously reset by the external reset signal (RST). After a reset, all bits in the registers are set to zero. The processor's main state machine is put into the IDLE state, maskable interrupts are masked and all external control signals go to an inactive state.

The processor stays in this state as long as a reset condition occurs. After reset (when signal RST changes to Low), the CPU starts executing an instruction fetch. The first instruction is fetched from address 0000h.

Interrupts

The TSK80x microcontroller has two interrupt request inputs – INT (maskable interrupt) and NMI (non-maskable interrupt). Interrupt service processing depends on the type of interrupt that was detected.

Non-maskable interrupt

The non-maskable interrupts cannot be disabled by program control and therefore will be accepted at all times by the CPU. They have a higher priority than maskable interrupts and are serviced first when both NMI and INT active signals are received at the same time.

After an interrupt is detected, the processor makes a special machine cycle (shown in Figure 7). This cycle is similar to the FETCH cycle but no data is read. The CPU then pushes the contents of the Program Counter onto the stack and makes a jump to address 0066h (the first instruction of the non-maskable service routine should be placed at this location).

In addition, maskable interrupts are disabled, preventing any further interrupts during the servicing of the non-maskable interrupt. The enable status of maskable interrupts is stored in the Interrupt Enable flip-flop IFF1. When a non-maskable interrupt is accepted, this flip-flop is reset (0), disabling maskable interrupts from being accepted. Its previous state (enabled or disabled) is stored however, enabling the interrupt state prior to the non-maskable interrupt to be restored after the current interrupt has been serviced. The state of the maskable interrupt is stored by copying the contents of interrupt enable flip-flop IFF1 into IFF2.

Note: The non-maskable interrupt input is edge-triggered. The signal is tested on the rising edge of the last cycle during the current instruction.

The NMI signal is an asynchronous input. If signal NMI changes value (from 0 to 1) before the last clock period of an instruction cycle, for the interrupt to be recognized the rising edge must be t_{set} before the rising edge of that last clock (t_{set} is equal to the delay time on one level logic). Otherwise the next instruction will be executed and the interrupt will be handled after its completion.

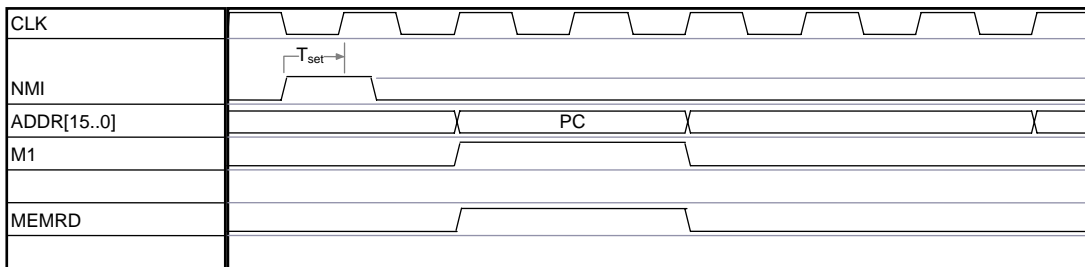


Figure 7. Non-maskable interrupt

Note: PC - instruction address (contents of Program Counter (PC))

Maskable interrupts

The maskable interrupt input (INT) is level-sensitive and is tested at the rising edge of the last clock cycle of any instruction. If it is High, the interrupt service is enabled and, providing interrupts have been enabled and there is no non-maskable interrupt request or bus request, the special interrupt acknowledge cycle begins.

Note that after a reset of the processor, interrupts are, by default, disabled. The EI instruction should be used to enable interrupts. After the EI instruction is executed, any interrupt request that is waiting for acknowledgement will not be accepted until the instruction after the EI instruction has been executed.

The interrupt acknowledge cycle is similar to the FETCH cycle, with the difference being that signal SFRRD becomes active rather than MEMRD and additional wait cycles are inserted. These additional wait cycles force the processor to wait before reading data and therefore enables an external device to properly synchronize. Figure 8 shows a maskable interrupt request/ acknowledge cycle.

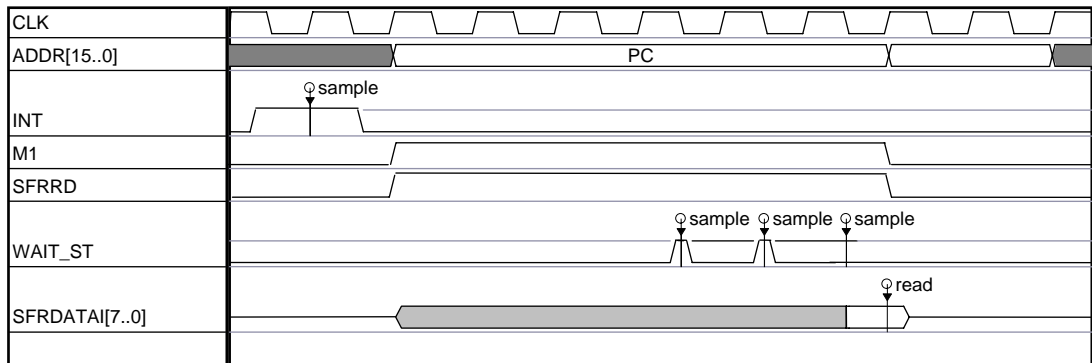


Figure 8. Maskable interrupt request and acknowledge cycle

Note: PC - instruction address (contents of Program Counter PC)
 read - point at which the data is read into the CPU
 sample - point at which signal is tested (SFRRD, WAIT_ST)

When an interrupt request is detected, the next cycle starts as a normal FETCH cycle (the contents of the PC are put onto the address bus and M1 is activated).

On the falling edge of the second clock cycle after M1 goes active, the SFRRD signal changes to High and the CPU waits on data. On the falling edge of the next clock cycle, signal WAIT_ST is tested and if it is active (High), the processor stays in this same state until the WAIT_ST signal is changed to Low.

Each accepted interrupt request causes a clear of the IFF1 flip-flop - to disable interrupts.

On the rising edge of the next clock cycle after WAIT_ST goes Low, the CPU reads data from the data bus. How this information will be used depends on the interrupt mode. This mode is changed under program control and, by default, after a reset of the processor it is set in mode 0. The TSK80x has three maskable interrupt modes described in the following section.

Note: Maskable interrupts are requested by a High level on signal INT, as opposed to a rising edge on the NMI signal, used to request a non-maskable interrupt.

Maskable interrupt modes

The interrupt request/acknowledge cycle is the same for every mode. The difference is how the data is read during this cycle.

MODE 0 – In this mode, the peripheral device requesting the interrupt can place any instruction on the data bus. For single byte instructions, the processor reads the instruction during acknowledgement of the interrupt request and executes it in the normal way. In general, it is a restart instruction, but all other instructions can be used.

For instructions that require more than one byte of data (e.g. CALL), the external device must disconnect memory and place the needed data onto the data bus in the correct time period (the processor reads only one byte during the interrupt request/acknowledge cycle; all other bytes of data are read in accordance with the processor's normal data memory read cycle).

MODE 1 – In this mode, a normal interrupt acknowledge cycle is made, but data put onto the data bus is not read by the processor, it is ignored. This mode is very similar to that of a non-maskable interrupt request/acknowledge but the CPU jumps to another address (0038h) for the first instruction of the service routine.

MODE 2 – This is the most flexible interrupt mode. In this mode, during the interrupt cycle, the processor reads a byte of data from the data bus and treats it not as an instruction (like in MODE 0), but as the low order byte of the address bus. The high order byte of the address bus is loaded from the processor's Interrupt register (I). This address points to the location in memory where the first instruction of the interrupt service routine is stored (see Figure 9).

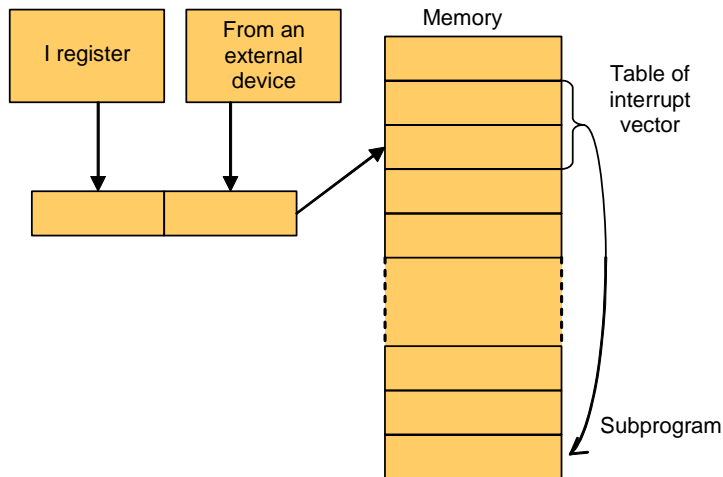


Figure 9. Call interrupt routine in mode 2.

This method of address construction allows different interrupt service routines for different devices to be used, or many interrupt service routines for one device depending on the situation. Additionally, all interrupt service routines can be placed in any memory location.

On-Chip Debugging

The debug-enabled version of the microcontroller (TSK80A_D) provides the following set of additional functional features that facilitate real-time debugging of the microcontroller:

- Reset, Go, Halt processor control
- Single or multi-step debugging
- Read-write access for internal processor registers including IX, IY, SP and PC
- Read-write access for memory and I/O space
- Hardware execution breakpoints (configurable number of breakpoint address registers and breakpoint data registers) allows trigger address and data for memory and I/O space.
- Hardware triggers can be set on an address range and/or data with masking bits
- Unlimited software breakpoints.

Adding debug functionality to the standard core

The debug functionality of the TSK80A_D is provided through the use of an On-Chip Debug System unit (OCDS). The simplified block diagram of Figure 10 shows the connection between this unit and the standard TSK80A core.

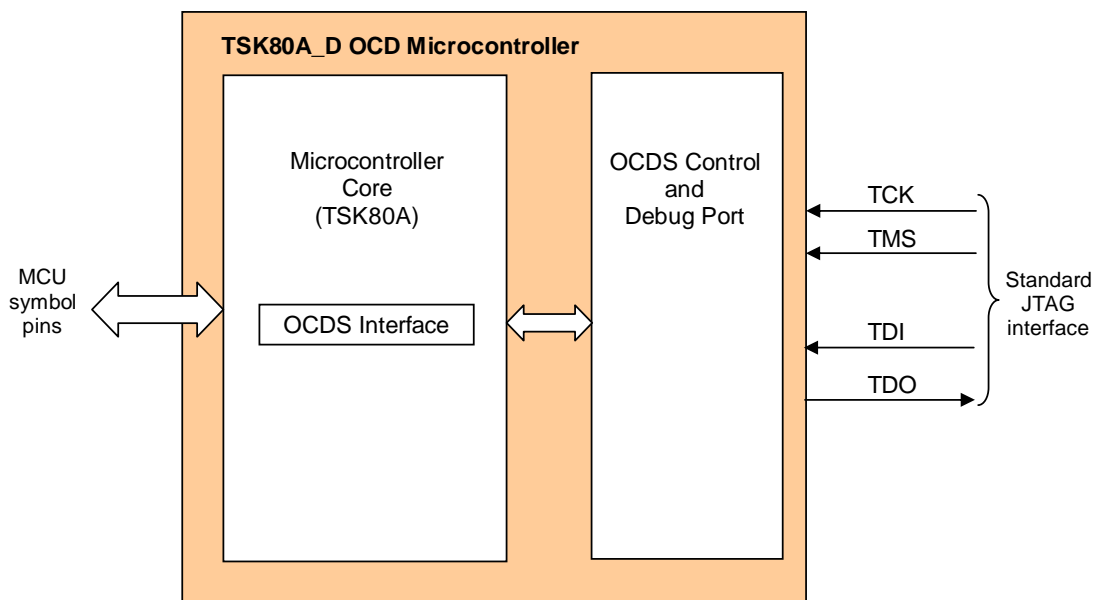


Figure 10. Simplified TSK80A_D block diagram

The host computer is connected to the target core using the IEEE 1149.1 (JTAG) standard interface. This is the physical interface, providing connection to physical pins of the FPGA device in which the core has been embedded.

The Nexus 5001 standard is used as the protocol for communications between the host and all devices that are debug-enabled with respect to this protocol. This includes all OCD-version microcontrollers, as well as other Nexus-compliant devices such as frequency generators, logic analyzers, counters, etc.

All such devices are connected in a chain – the Soft Devices chain – which is determined when the design has been implemented within the target FPGA device and presents in the **Devices** view (Figure 11). It is not a physical chain, in the sense that you can see no external wiring – the connections required between the Nexus-enabled devices are made internal to the FPGA itself.

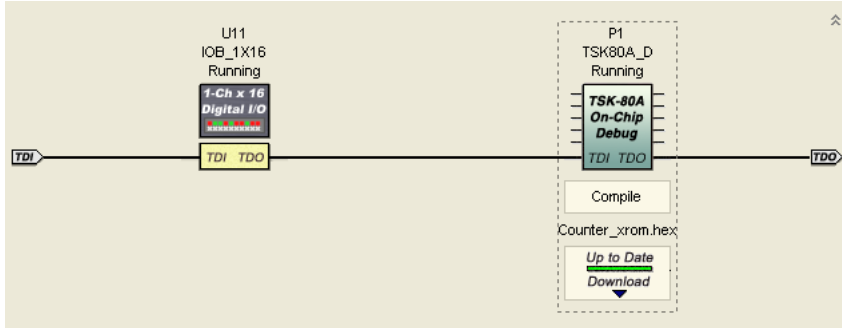


Figure 11. Nexus-enabled microcontroller appearing in the Soft Devices chain

For microcontrollers such as the TSK80A_D, the Nexus protocol enables you to debug the core through communication with the OCDS Unit.

Accessing the debug environment

Debugging of the embedded code within an OCD-version microcontroller is carried out by starting a debug session. Prior to starting the session, you must ensure that the design, including one or more OCD-version microcontrollers and their respective embedded code, has been downloaded to the target physical FPGA device.

To start a debug session for the embedded code of a specific microcontroller in the design, simply right-click on the icon for that microcontroller, in the Soft Devices region of the view, and choose the **Debug** command from the pop-up menu that appears. Alternatively, click on the icon for the microcontroller (to focus it) and choose **Processors » Pn » Debug** from the main menus, where n corresponds to the number for the processor in the Soft Devices chain.

The embedded project for the software running in the processor will initially be recompiled and the debug session will commence. The relevant source code document (either Assembly or C) will be opened and the current execution point will be set to the first line of executable code (see Figure 12).

Note: You can have multiple debug sessions running simultaneously – one per embedded software project associated with a microcontroller in the Soft Devices chain.

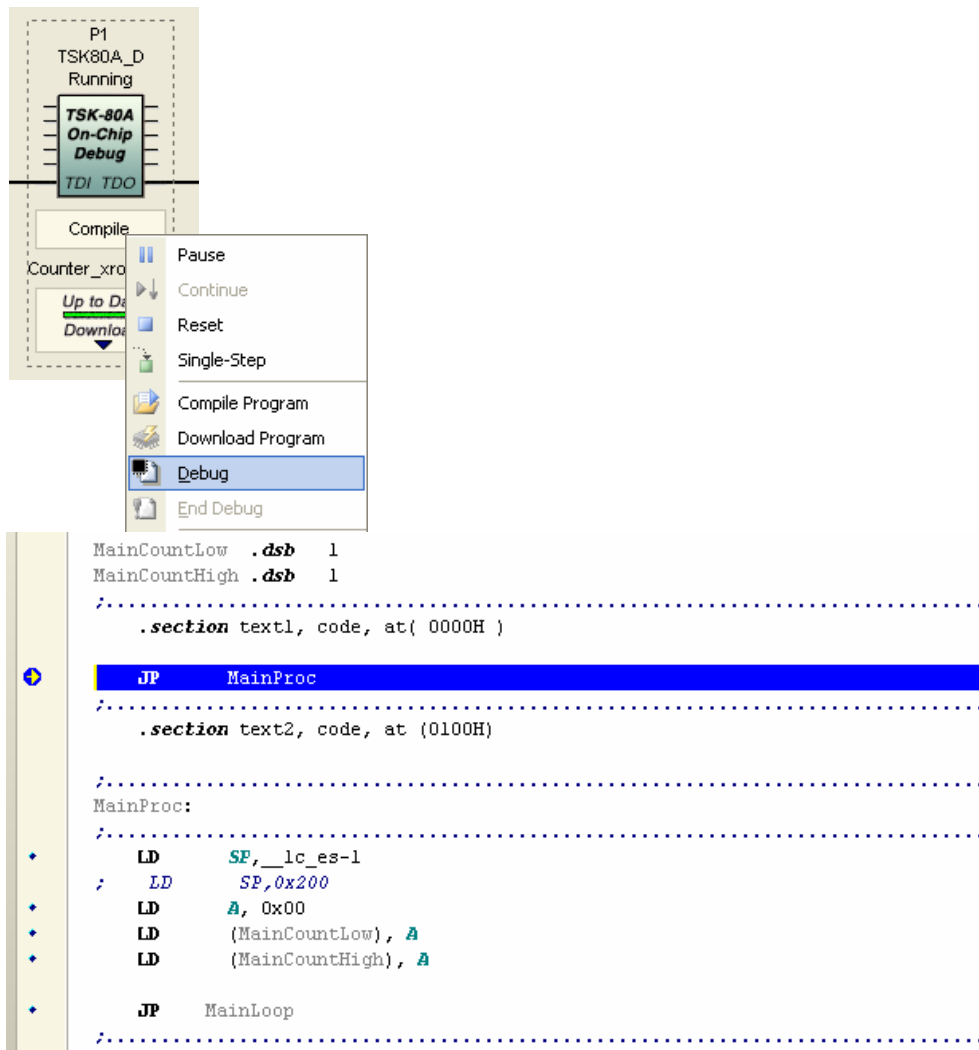


Figure 12. Starting an embedded code debug session.

The debug environment offers the full suite of tools you would expect to see in order to efficiently debug the embedded code. These features include:

- Setting Breakpoints
- Adding Watches
- Stepping into and over at both the source (*.c) and instruction (*.asm) level
- Reset, Run and Halt code execution
- Run to cursor

All of these and other feature commands can be accessed from the **Debug** menu or the associated **Debug** toolbar.

Various workspace panels are accessible in the debug environment, allowing you to view/control code-specific features, such as Breakpoints, Watches and Local variables, as well as information specific to the microcontroller in which the code is running, such as memory spaces and registers.

These panels can be accessed from the **View » Workspace Panels » Embedded** sub menu, or by clicking on the **Embedded** button at the bottom of the application window and choosing the required panel from the subsequent pop-up menu.

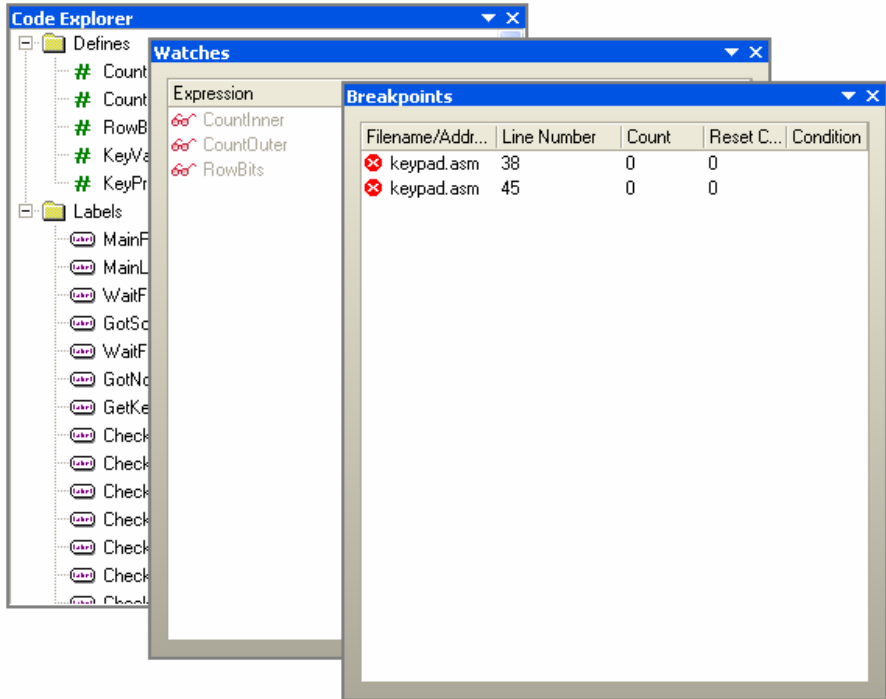


Figure 13. Workspace panels offering code-specific information and controls

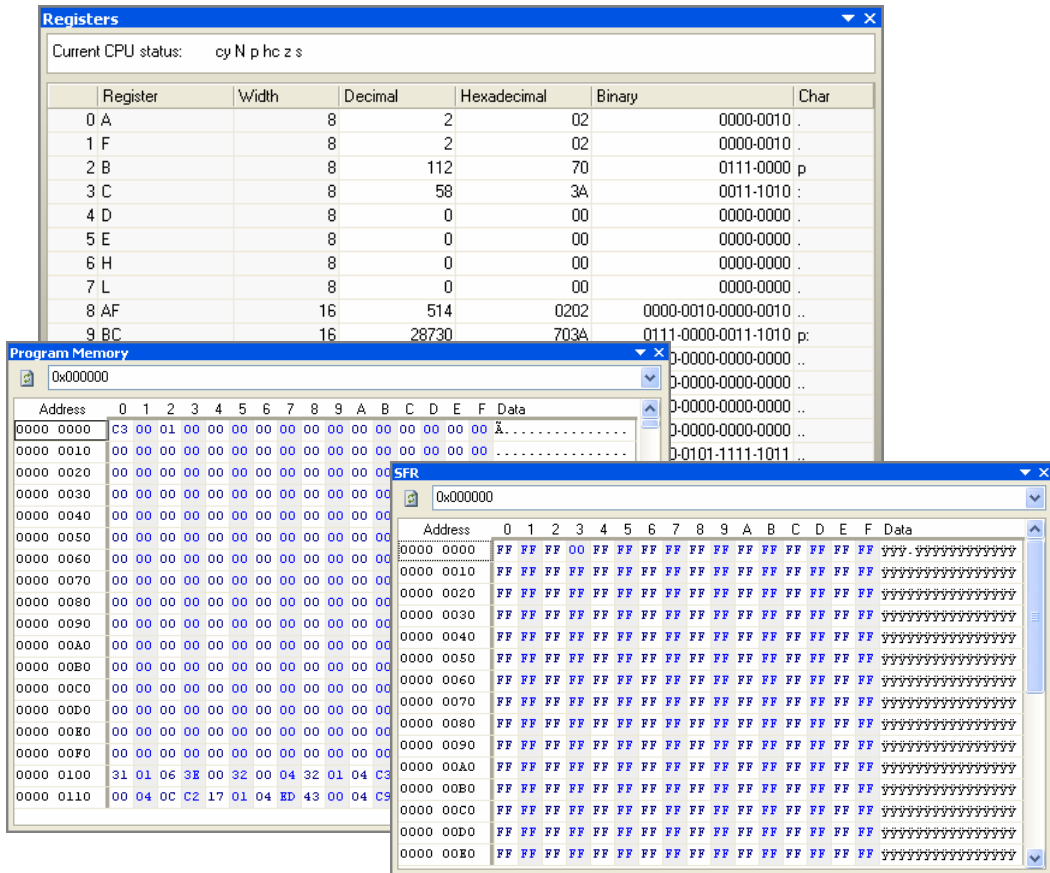


Figure 14. Workspace panels offering information specific to the parent processor.

Full-feature debugging is of course enjoyed at the source code level – from within the source code file itself. To a lesser extent, debugging can also be carried out from a dedicated debug panel for the processor. To access² this panel, first double-click on the icon representing the microcontroller to be debugged, in the **Soft Devices** region of the view. The *Instrument Rack – Soft Devices* panel will appear, with the chosen processor instrument added to the rack (Figure 15).

² The debug panels for each of the debug-enabled microcontrollers are standard panels and, as such, can be readily accessed from the **View » Workspace Panels » Instruments** sub menu, or by clicking on the **Instruments** button at the bottom of the application window and choosing the required panel – for the processor you wish to debug – from the subsequent pop-up menu.

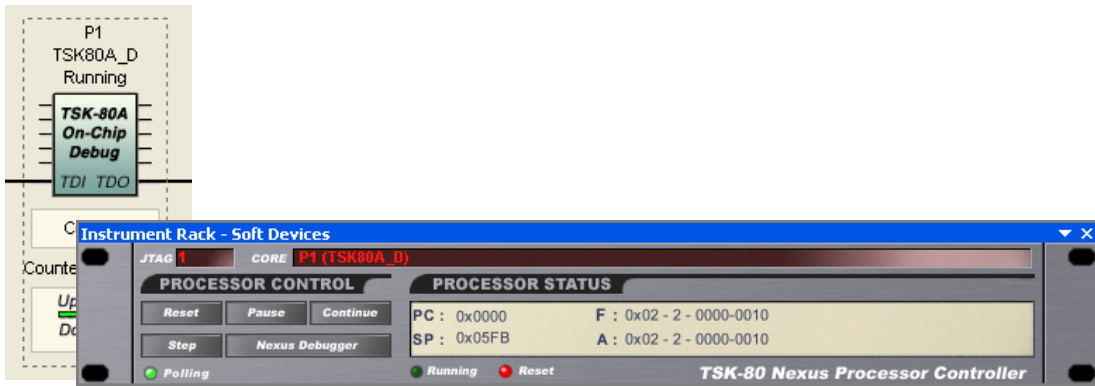





Figure 15. Accessing debug features from the microcontroller's instrument panel

Note: Each core microcontroller that you have included in the design will appear, when double-clicked, as an Instrument in the rack (along with any other Nexus-enabled devices).

The **Nexus Debugger** button provides access to the associated debug panel (Figure 16), which in turn allows you to interrogate and to a lighter extent control, debugging of the processor and its embedded code, notably with respect to the registers and memory.

One key feature of the panel is that it enables you to specify (and therefore change) the embedded code (HEX file) that is downloaded to the microcontroller, quickly and efficiently.

-  For more information on the content and use of processor debug panels, press **F1** when the cursor is over one of these panels.
-  For further information regarding the use of the embedded tools for the TSK80x, see the *Using the TSK80x Embedded Tools* guide.
-  For comprehensive information with respect to the embedded tools available for the TSK80x, see the *TSK80x Embedded Tools Reference*.

TSK80A_D - P1

Processor Registers

Name	Width	Hex	Binary	Signed	Unsigned	Char
A	8	0	0000-0000	0	0	.
F	8	2	0000-0010	2	2	.
PC	16	138	0000-0001-0011-1000	312	312	.8
SP	16	5FB	0000-0101-1111-1011	1,531	1,531	..
AF	16	2	0000-0000-0000-0010	2	2	..
HL	16	0	0000-0000-0000-0000	0	0	..
B	8	2A	0010-1010	42	42	*
C	8	4	0000-0100	4	4	.
D	8	0	0000-0000	0	0	.
E	8	0	0000-0000	0	0	.
H	8	0	0000-0000	0	0	.
L	8	0	0000-0000	0	0	.
A'	8	0	0000-0000	0	0	.

Refresh

```

code:0x0138  05      DEC  B
code:0x0139  c2 38 01    JP   NZ,code:0x0138
code:0x013c  0d          DEC  C
code:0x013d  c2 36 01    JP   NZ,code:0x0136
    
```

Code SFR Data

Refresh [Dropdown] Auto [Dropdown] 0x000000 [Dropdown]

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Data	
0000 0000	c3	00	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000 0010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000 0020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000 0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000 0040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000 0050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000 0060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000 0070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000 0080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000 0090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Reset Pause Continue Single Step Run Steps 1000

Figure 16. Processor debugging using the associated processor debug panel

Instruction set

All TSK80x instructions are binary code compatible.

Table 2. Instruction operand descriptions

Mnemonic	Description
r / r'	Working register A, B, C, D, E, H, L / A', B', C', D', E', H', L'
b	single bit
n	Any 8-bit immediate data
d	Signed 8-bit displacement, from an Index register or Program Counter
nn	Any 16-bit immediate data
pp	Pointer to a 16-bit register BC, DE, HL, IX, IY, AF or SP. Note, in the concatenation of the A and F registers, the Accumulator is used as the MSB
e	relative jump (8 bits wide)

Instruction set – functional groupings

Table 3. 8-bit arithmetic operations

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
ADC A, (HL)	Add byte, from memory location selected by contents of HL register, to Accumulator with carry flag	7	1
ADC A, (IX+d) ADC A, (IY+d)	Add byte, from memory location selected by sum of the contents of an Index register (IX or IY) and a signed 8-bit displacement, to Accumulator with carry flag	9	3
ADC A, n	Add immediate data to Accumulator with carry flag	7	2
ADC A, r	Add register to Accumulator with carry flag	5	1
ADD A, (HL)	Add byte, from memory location selected by contents of HL register, to Accumulator	7	1
ADD A, (IX+d) ADD A, (IY+d)	Add byte, from memory location selected by sum of the contents of an Index register (IX or IY) and a signed 8-bit displacement, to Accumulator	9	3
ADD A, n	Add immediate data to Accumulator	7	2
ADD A, r	Add register to Accumulator	5	1

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
DEC (HL)	Decrement byte location in memory selected by contents of register HL	7	1
DEC (IX+d) DEC (IY+d)	Decrement byte location in memory selected by sum of the contents of an Index register (IX or IY) and an 8-bit signed displacement	9	3
DEC r	Decrement register	5	1
INC (HL)	Increment byte location in memory selected by contents of register HL	7	1
INC (IX+d) INC (IY+d)	Increment byte location in memory selected by sum of the contents of an Index register (IX or IY) and an 8-bit signed displacement	9	3
INC r	Increment register	5	1
SBC A, (HL)	Subtract byte, from memory location selected by sum of the contents of HL register, from Accumulator with carry flag	7	1
SBC A, (IX+d) SBC A, (IY+d)	Subtract byte, from memory location selected by sum of the contents of an Index register (IX or IY) and a signed 8-bit displacement, from Accumulator with carry flag	9	2
SBC A, n	Subtract immediate data from Accumulator with carry flag	7	2
SBC A, r	Subtract register from Accumulator with carry flag	5	1
SUB A, (HL)	Subtract byte, from memory location selected by contents of HL register, from Accumulator	7	1
SUB A, (IX+d) SUB A, (IY+d)	Subtract byte, from memory location selected by sum of the contents of an Index register (IX or IY) and a signed 8-bit displacement, from Accumulator	9	3
SUB A, n	Subtract immediate data from Accumulator	7	2
SUB A, r	Subtract register from Accumulator	5	1

Table 4. 16-bit arithmetic operations

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
ADC HL, pp	Add contents of 16-bit register pp to HL register with carry flag	7	2
ADD HL, pp	Add contents of 16-bit register pp to HL register	6	1
ADD IX, pp ADD IY, pp	Add to Index register (IX or IY) the contents of the 16-bit register pp	7	2
DEC IX DEC IY	Decrement contents of Index register (IX or IY)	7	2
DEC pp	Decrement contents of 16-bit register pp	6	1
INC IX INC IY	Increment contents of Index register (IX or IY)	7	2
INC pp	Increment contents of 16-bit register pp	6	1
SBC HL, pp	Subtract contents of 16-bit register pp from register HL with carry flag	7	2

Table 5. Logic operations

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
AND A, (HL)	Logically AND the byte, from memory location selected by the contents of the HL register, to Accumulator	7	1
AND A, (IX+d) AND A, (IY+d)	Logically AND the byte, from memory location selected by sum of the contents of an Index register (IX or IY) and a signed 8-bit displacement, to Accumulator	9	3
AND A, n	Logically AND immediate data to Accumulator	7	2
AND r	Logically AND register to Accumulator	5	1
CP A, (HL)	Compare byte, from memory location selected by the contents of the HL register, to Accumulator	7	1
CP A, (IX+d) CP A, (IY+d)	Compare byte, from memory location selected by sum of the contents of an Index register (IX or IY) and a signed 8-bit displacement, to Accumulator	9	3

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
CP A, n	Compare immediate data to Accumulator	7	2
CP A, r	Compare contents of register to Accumulator	5	1
OR A, (HL)	Logically OR the byte, from memory location selected by the contents of the HL register, to Accumulator	7	1
OR A, (IX+d) OR A, (IY+d)	Logically OR the byte, from memory location selected by sum of the contents of an Index register (IX or IY) and a signed 8-bit displacement, to Accumulator	9	3
OR A, n	Logically OR immediate data to Accumulator	7	2
OR A, r	Logically OR register to Accumulator	5	1
XOR A, (HL)	Logically Exclusive OR the byte, from memory location selected by the contents of the HL register, to Accumulator	7	1
XOR A, (IX+d) XOR A, (IY+d)	Logically Exclusive OR the byte, from memory location selected by sum of the contents of an Index register (IX or IY) and a signed 8-bit displacement, to Accumulator	9	3
XOR A, n	Logically Exclusive OR immediate data to Accumulator	7	2
XOR A, r	Logically Exclusive OR register to Accumulator	5	1

Table 6. 8-bit load instructions

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
LD (BC), A	Load value in the Accumulator into the memory location selected by the BC register	4	1
LD (DE), A	Load value in the Accumulator into the memory location selected by the DE register	4	1
LD (HL), n	Load immediate data to memory location selected by the contents of the HL register	5	2
LD (HL), r	Load byte from register to memory location selected by the contents of the HL register	4	1

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
LD (IX+d), n LD (IY+d), n	Load immediate data to memory location selected by sum of the contents of an Index register (IX or IY) and a signed 8-bit displacement	7	4
LD (IX+d), r LD (IY+d), r	Load byte from register to memory location, selected by sum of the contents of an Index register (IX or IY) and a signed 8-bit displacement	6	3
LD (nn), A	Load value in the Accumulator into the memory location selected by direct address nn in the instruction itself	6	3
LD A, (BC)	Load byte from memory location selected by contents of the BC register	4	1
LD A, (DE)	Load byte from memory location selected by contents of the DE register	4	1
LD A, (nn)	Load byte from memory location selected by direct address nn into the Accumulator	6	3
LD A, I	Load contents of Interrupt register into Accumulator	4	2
LD I, A	Load contents of Accumulator into the Interrupt register	4	2
LD r, (HL)	Load byte, from memory location selected by the contents of the HL register, to register	4	1
LD r, (IX+d) LD r, (IY+d)	Load byte, from memory location selected by sum of the contents of an Index register (IX or IY) and a signed 8-bit displacement, to register	6	3
LD r, n	Load immediate data to register	4	2
LD r1, r2	Load data from register r2 to register r1	3	1

Table 7. 16-bit load instructions

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
LD (nn), pp	The contents of register pp are loaded into memory location described by direct address nn	8	4
LD (nn), HL	The contents of register HL are loaded into memory location described by direct address nn	7	3

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
LD (nn), IX LD (nn), IY	The contents of Index register IX or IY are loaded into memory location described by direct address nn	8	4
LD pp, (nn)	The contents of memory location described by direct address nn are loaded into register pp	8	4
LD pp, nn	Load immediate data into register pp	5	3
LD HL, (nn)	The contents of memory location described by direct address nn are loaded into register HL	5	3
LD IX, (nn) LD IY, (nn)	The contents of memory location described by direct address nn are loaded into Index register (IX or IY)	8	4
LD IX, nn LD IY, nn	Load immediate data into Index register (IX or IY)	6	4
LD SP, IX LD SP, IY	The contents of Index register IX or IY are loaded into the Stack Pointer register	4	2
LD SP,HL	The contents of register HL are loaded into the Stack Pointer register	4	1
POP IX POP IY	Two bytes are “popped” from the stack in external memory and loaded into index register IX or IY	6	2
POP pp	Two bytes are “popped” from the stack in external memory and loaded into 16-bit register pp	5	1
PUSH IX PUSH IY	The contents of the Index register (IX or IY) are “pushed” into the stack in external memory	6	2
PUSH pp	The contents of 16-bit register pp are “pushed” into the stack in external memory	5	1

Table 8. General-purpose arithmetic and control instructions

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
CCF	Invert carry flag in F register	3	1
CPL	Complement Accumulator	3	1

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
DAA	Convert Accumulator into packed BCD	5	1
DI	Disable interrupt	3	1
EI	Enable interrupt	3	1
HALT	Halt the microprocessor	3	1
IM 0	Set interrupt in mode 0	4	2
IM 1	Set interrupt in mode 1	4	2
IM 2	Set interrupt in mode 2	4	2
NEG	The contents of Accumulator are negated	4	2
NOP	NO operation	3	1
SCF	Set carry flag in F register	3	1

Table 9. Jump instructions

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
DJNZ e	Decrement register B, if B=0 fetch next instruction, else jump relative to PC at e (e can be plus or minus)	8(13)*	2
JP C, nn	Jump to address nn if flag C is set	6	3
JP HL	Jump to address stored in HL register	3	1
JP IX JP IY	Jump to address stored in Index register (IX or IY)	4	2
JP M, nn	Jump to address nn if flag S is set	6	3
JP NC, nn	Jump to address nn if flag C is reset	6	3
JP nn	Jump to address nn	6	3
JP NZ, nn	Jump to address nn if flag Z is reset	6	3
JP P, nn	Jump to address nn if flag S is reset	6	3
JP PE, nn	Jump to address nn if flag P/V is set	6	3
JP PO, nn	Jump to address nn if flag P/V is reset	6	3

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
JP Z, nn	Jump to address nn if flag Z is set	6	3
JR C, e	Jump relative to PC at e (e can be plus or minus) if carry flag is set	5(7)*	2
JR e	Jump relative to PC at e (e can be plus or minus)	5	2
JR NC, e	Jump relative to PC at e (e can be plus or minus) if carry flag is reset	5(7)*	2
JR NZ, e	Jump relative to PC at e (e can be plus or minus) if flag Z is reset	5(7)*	2
JR Z, e	Jump relative to PC at e (e can be plus or minus) if flag Z is set	5(7)*	2

* The number of clock cycles is shown in parentheses for cases when the condition is not true.

Table 10. Call and return instructions

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
CALL C, nn	Call to subroutine at address nn if Carry flag is set	7(3)*	3
CALL M, nn	Call to subroutine at address nn if S flag is set	7(3)*	3
CALL NC, nn	Call to subroutine at address nn if Carry flag is reset	7(3)*	3
CALL nn	Call to subroutine at address nn	7	3
CALL NZ, nn	Call to subroutine at address nn if Z flag is reset	7(3)*	3
CALL P, nn	Call to subroutine at address nn if S flag is reset	7(3)*	3
CALL PE, nn	Call to subroutine at address nn if P/V flag is set	7(3)*	3
CALL PO, nn	Call to subroutine at address nn if P/V flag is reset	7(3)*	3
CALL Z, nn	Call to subroutine at address nn if Z flag is set	7(3)*	3
RET	Return from subroutine	6	1
RET C	Return from subroutine if Carry flag is set	6(1)*	1
RET M	Return from subroutine if S flag is set	6(1)*	1
RET NC	Return from subroutine if Carry flag is reset	6(1)*	1

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
RET NZ	Return from subroutine if Z flag is reset	6(1)*	1
RET P	Return from subroutine if S flag is reset	6(1)*	1
RET PE	Return from subroutine if P/V flag is set	6(1)*	1
RET PO	Return from subroutine if P/V flag is reset	6(1)*	1
RET Z	Return from subroutine if Z flag is set	6(1)*	1
RETI	Return from servicing a maskable interrupt	7	2
RETN	Return from servicing a non-maskable interrupt	7	2
RST 0h	Restart (call) to address 0000h	6	1
RST 10h	Restart (call) to address 0010h	6	1
RST 18h	Restart (call) to address 0018h	6	1
RST 20h	Restart (call) to address 0020h	6	1
RST 28h	Restart (call) to address 0028h	6	1
RST 30h	Restart (call) to address 0030h	6	1
RST 38h	Restart (call) to address 0038h	6	1
RST 8h	Restart (call) to address 0008h	6	1

* The number of clock cycles is shown in parentheses for cases when the condition is not true.

Table 11. Rotate and shift instructions

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
RL (HL)	Rotate left, one bit position through the Carry flag, the data located in memory at the address stored in the HL register	8	2
RL (IX+d) RL (IY+d)	Rotate left, one bit position through the Carry flag, the data at the memory location whose address is the sum of the contents of an Index register (IX or IY) and a signed 8-bit displacement d	11	4
RL r	Rotate left, one bit position through the Carry flag, the contents of register r	6	2

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
RLA	Rotate left, one bit position through the Carry flag, the contents of the Accumulator	5	1
RLC (HL)	Rotate left, one bit position with Carry flag, the data located in memory at the address stored in the HL register	8	2
RLC (IX+d) RLC (IY+d)	Rotate left, one bit position with Carry flag, the data at the memory location whose address is the sum of the contents of an Index register (IX or IY) and a signed 8-bit displacement d	11	4
RLC r	Rotate left, one bit position with Carry flag, the contents of register r	6	2
RLCA	Rotate left, one bit position with Carry flag, the contents of the Accumulator	5	1
RLD	Rotate left, four bit positions, the byte of data at the memory location addressed by the contents of the HL register, through the low order nibble of the Accumulator	8	2
RR (HL)	Rotate right, one bit position through the Carry flag, the data located in memory at the address stored in the HL register	8	2
RR (IX+d) RR (IY+d)	Rotate right, one bit position through the Carry flag, the data at the memory location whose address is the sum of the contents of an Index register (IX or IY) and a signed 8-bit displacement d	11	4
RR r	Rotate right, one bit position through the Carry flag, the contents of register r	6	2
RRA	Rotate right, one bit position through the Carry flag, the contents of the Accumulator	5	1
RRC (HL)	Rotate right, one bit position with Carry flag, the data located in memory at the address stored in the HL register	8	2
RRC (IX+d) RRC (IY+d)	Rotate right, one bit position with Carry flag, the data at the memory location whose address is the sum of the contents of an Index register (IX or IY) and a signed 8-bit displacement d	11	4
RRC r	Rotate right, one bit position with Carry flag, the contents of register r	6	2

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
RRCA	Rotate right, one bit position with Carry flag, the contents of the Accumulator	5	1
RRD	Rotate right, four bit positions, the contents of the HL register, through the low order nibble of the Accumulator	8	2
SLA (HL)	Shift arithmetically left, one bit position, the data located in memory at the address stored in the HL register	8	2
SLA (IX+d) SLA (IY+d)	Shift arithmetically left, one bit position, the data at the memory location whose address is the sum of the contents of an Index register (IX or IY) and a signed 8-bit displacement d	11	4
SLA r	Shift arithmetically left, one bit position, the contents of the register r	6	2
SRA (HL)	Shift arithmetically right, one bit position, the data located in memory at the address stored in the HL register	8	2
SRA (IX+d) SRA (IY+d)	Shift arithmetically right, one bit position, the data at the memory location whose address is the sum of the contents of an Index register (IX or IY) and a signed 8-bit displacement d	11	4
SRA r	Shift arithmetically right, one bit position, the contents of the register r	6	2
SRL (HL)	Shift logically right, one bit position, the data located in memory at the address stored in the HL register	8	2
SRL (IX+d) SRL (IY+d)	Shift logically right, one bit position, the data at the memory location whose address is the sum of the contents of an Index register (IX or IY) and a signed 8-bit displacement d	11	4
SRL r	Shift logically right, one bit position, the contents of the register r	6	2

Table 12. Input and output instructions

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
IN A, (n)	One byte of data from the I/O port selected by the contents at the address n is loaded into the Accumulator	5	2
IN r, (C)	One byte of data from the I/O port selected by the contents of register C, is loaded into register r.	5	2
IND	One byte of data is read from the I/O port selected by the contents of register C and placed into memory at the location specified by the contents of the HL register. Both the HL and B registers are then decremented. Register B is used as a byte counter.	8	2
INDR	Functions the same as the IND instruction but at the end, tests if B=0. If the test returns FALSE, the instruction is repeated. If TRUE, the instruction finishes.	5	2
INI	One byte of data is read from the I/O port selected by the contents of register C and placed into memory at the location specified by the contents of the HL register. The HL register is then incremented and register B is decremented. Register B is used as a byte counter.	8	2
INIR	Functions the same as the INI instruction but at the end, tests if B=0. If the test returns FALSE, the instruction is repeated. If TRUE, the instruction finishes.	8	2
OTDR	Functions the same as the OUTD instruction but at the end, it tests the contents of register B. If it is 0, the instruction is finished, otherwise it repeats.	8	2
OTIR	Functions the same as the OUTI instruction but at the end, it tests the contents of register B. If it is 0, the instruction is finished, otherwise it repeats.	8	2
OUT (C), r	The byte of data from register r is written to the external peripheral device through the port selected by the contents of register C.	5	2
OUT (n), A	The byte of data from the Accumulator is written to the external peripheral device through the port selected by the contents at address n.	5	2

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
OUTD	The byte of data contained in memory, at the address specified by the contents of the HL register, is written to the external peripheral device through the port selected by the contents of register C. Both register B (byte counter) and the HL register are decremented.	8	2
OUTI	The byte of data contained in memory, at the address specified by the contents of the HL register, is written to the external peripheral device through the port selected by the contents of register C. Register B (byte counter) is decremented and the HL register is incremented.	8	2

* The number of clock cycles is shown in parentheses for cases when the condition is not true.

Table 13. Exchange, block transfer and search instructions

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
CPD	Compares the contents of the memory location specified by the contents of the HL register, with the contents of the Accumulator. Depending on the result, the relevant status flag is set/reset. Both the HL and BC registers are decremented.	7	2
CPDR	Functions the same as the CPD instruction, but is repeated until BC=0 or A equals (HL).	7(16)*	2
CPI	Compares the contents of the memory location specified by the contents of the HL register, with the contents of the Accumulator. Depending on the result, the relevant status flag is set/reset. The HL register is incremented. The BC register (byte counter) is decremented.	7	2
CPIR	Functions the same as the CPI instruction, but is repeated until BC=0 or A equals (HL).	7(16)*	2
EX (SP), HL	Exchange the two byte data value in register HL with two bytes of data on top of the Stack	7	1
EX (SP), IX EX (SP), IY	Exchange contents of register IX or IY with the two bytes of data obtained from the memory addresses contained in the top two levels of the Stack.	8	2

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
EX AF, AF'	Exchange contents of registers AF (basic) and AF' (alternative)	3	1
EX DE, HL	Exchange the two byte data values in the basic general purpose registers DE and HL	3	1
EXX	Exchange the two byte data value in the basic general purpose registers BC, DE and HL with the two byte data value in the alternative general purpose registers BC', DE' and HL'	3	1
LDD	Transfer a byte of data from the memory location addressed by the contents of the HL register, to the memory location addressed by the contents of the DE register. The DE, HL and BC (byte counter) registers are all decremented.	9	2
LDDR	Functions the same as the LDD instruction but at the end, it tests the contents of register BC. If it is 0, the instruction is finished, otherwise it repeats.	9(16)*	2
LDI	Transfer a byte of data from the memory location addressed by the contents of the HL register, to the memory location addressed by the contents of the DE register. The DE and HL registers are incremented; the BC register (byte counter) is decremented.	9	2
LDIR	Functions the same as the LDI instruction but at the end, it tests the contents of register BC. If it is 0, the instruction is finished, otherwise it repeats.	9(16)*	2

* The number of clock cycles is shown in parentheses for cases when the condition is not true.

Table 14. Bit manipulation instructions

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
BIT b, (HL)	Test bit b in the memory location specified by the contents of the HL register and set the Z flag accordingly.	7	2

Mnemonic	Description	Total Clock Cycles for execution	Width (in bytes)
BIT b, (IX+d) BIT b, (IY+d)	Test bit b in the memory location specified by the sum of the contents of the Index register (IX or IY) and the signed 8-bit displacement d. Set the Z flag accordingly.	11	4
BIT b, r	Test bit b in register r and set the Z flag accordingly.	5	2
RES b, (HL)	Reset bit b in the memory location specified by the contents of the HL register.	8	2
RES b, (IX+d) RES b, (IY+d)	Reset bit b in the memory location specified by the sum of the contents of the Index register (IX or IY) and the signed 8-bit displacement d.	11	4
RES b, r	Reset bit b in register r	6	2
SET b, (HL)	Set bit b in the memory location specified by the contents of the HL register.	8	2
SET b, (IX+d) SET b, (IY+d)	Set bit b in the memory location specified by the sum of the contents of the Index register (IX or IY) and the signed 8-bit displacement d.	11	4
SET b, r	Set bit b in register r	6	2

Hexadecimal ordered instructions

Table 15. Instruction set without prefix

Opcode	Mnemonic	Opcode	Mnemonic
00h	NOP	10h	DJNZ e
01h	LD BC, nn	11h	LD DE, nn
02h	LD (BC), A	12h	LD (DE), A
03h	INC BC	13h	INC DE
04h	INC B	14h	INC D
05h	DEC B	15h	DEC D
06h	LD B, n	16h	LD D, n
07h	RLCA	17h	RLA
08h	EX AF, AF'	18h	JR n

Opcode	Mnemonic	Opcode	Mnemonic
09h	ADD HL, BC	19h	ADD HL, DE
0Ah	LD A, (BC)	1Ah	LD A, (DE)
0Bh	DEC BC	1Bh	DEC DE
0Ch	INC C	1Ch	INC E
0Dh	DEC C	1Dh	DEC E
0Eh	LD C, n	1Eh	LD E, n
0Fh	RRCA	1Fh	RRA
20h	JR NZ, n	30h	JR NC, n
21h	LD HL, nn	31h	LD SP, nn
22h	LD (nn) HL	32h	LD (nn), A
23h	INC HL	33h	INC SP
24h	INC H	34h	INC (HL)
25h	DEC H	35h	DEC (HL)
26h	LD H, n	36h	LD (HL), n
27h	DAA	37h	SCF
28h	JR Z, n	38h	JR C, n
29h	ADD HL, HL	39h	ADD HL, SP
2Ah	LD HL, (nn)	3Ah	LD A, (nn)
2Bh	DEC HL	3Bh	DEC SP
2Ch	INC L	3Ch	INC A
2Dh	DEC L	3Dh	DEC A
2Eh	LD L, n	3Eh	LD A, n
2Fh	CPL	3Fh	CCF
40h	LD B, B	50h	LD D, B
41h	LD B, C	51h	LD D, C
42h	LD B, D	52h	LD D, D
43h	LD B, E	53h	LD D, E
44h	LD B, H	54h	LD D, H

Opcode	Mnemonic	Opcode	Mnemonic
45h	LD B, L	55h	LD D, L
46h	LD B, (HL)	56h	LD D, (HL)
47h	LD B, A	57h	LD D, A
48h	LD C, B	58h	LD E, B
49h	LD C, C	59h	LD E, C
4Ah	LD C, D	5Ah	LD E, D
4Bh	LD C, E	5Bh	LD E, E
4Ch	LD C, H	5Ch	LD E, H
4Dh	LD C, L	5Dh	LD E, L
4Eh	LD C, (HL)	5Eh	LD E, (HL)
4Fh	LD C, A	5Fh	LD E, A
60h	LD H, B	70h	LD (HL), B
61h	LD H, C	71h	LD (HL), C
62h	LD H, D	72h	LD (HL), D
63h	LD H, E	73h	LD (HL), E
64h	LD H, H (reserved in TSK80A_D)	74h	LD (HL), H
65h	LD H, L	75h	LD (HL), L
66h	LD H, (HL)	76h	HALT
67h	LD H, A	77h	LD (HL), A
68h	LD L, B	78h	LD A, B
69h	LD L, C	79h	LD A, C
6Ah	LD L, D	7Ah	LD A, D
6Bh	LD L, E	7Bh	LD A, E
6Ch	LD L, H	7Ch	LD A, H
6Dh	LD L, L	7Dh	LD A, L
6Eh	LD L, (HL)	7Eh	LD A, (HL)
6Fh	LD L, A	7Fh	LD A, A
80h	ADD A, B	90h	SUB B

Opcode	Mnemonic	Opcode	Mnemonic
81h	ADD A, C	91h	SUB C
82h	ADD A, D	92h	SUB D
83h	ADD A, E	93h	SUB E
84h	ADD A, H	94h	SUB H
85h	ADD A, H	95h	SUB L
86h	ADD A, (HL)	96h	SUB (HL)
87h	ADD A, A	97h	SBC A
88h	ADC A, B	98h	SBC B
89h	ADC A, C	99h	SBC C
8Ah	ADC A, D	9Ah	SBC D
8Bh	ADC A, E	9Bh	SBC E
8Ch	ADC A, H	9Ch	SBC H
8Dh	ADC A, H	9Dh	SBC L
8Eh	ADC A, (HL)	9Eh	SBC (HL)
8Fh	ADC A, A	9Fh	SBC A
A0h	AND B	B0h	OR B
A1h	AND C	B1h	OR C
A2h	AND D	B2h	OR D
A3h	AND E	B3h	OR E
A4h	AND H	B4h	OR H
A5h	AND L	B5h	OR L
A6h	AND (HL)	B6h	OR (HL)
A7h	AND A	B7h	OR A
A8h	XOR B	B8h	CP B
A9h	XOR C	B9h	CP C
AAh	XOR D	BAh	CP D
ABh	XOR E	BBh	CP E
ACh	XOR H	BCh	CP H

Opcode	Mnemonic	Opcode	Mnemonic
ADh	XOR L	BDh	CP L
AEh	XOR (HL)	BEh	CP (HL)
AFh	XOR A	BFh	CP A
C0h	RET NZ	D0h	RET NC
C1h	POP BC	D1h	POP DE
C2h	JP NZ, nn	D2h	JP NC, nn
C3h	JP nn	D3h	OUT (n), A
C4h	CALL NZ, nn	D4h	CALL NC, nn
C5h	PUSH BC	D5h	PUSH DE
C6h	ADD A,n	D6h	SUB n
C7h	RST 00	D7h	RST 10
C8h	RET Z	D8h	RET C
C9h	RET	D9h	EXX
CAh	JP Z, nn	DAh	JP C, nn
CBh	Prefix	DBh	IN A, (n)
CCh	CALL Z, nn	DCh	CALL C, nn
CDh	CALL nn	DDh	Prefix
CEh	ADC A, n	DEh	SBC A, n
CFh	RST 08	DFh	RST 18
E0h	RET PO	F0h	RET P
E1h	POP HL	F1h	POP AF
E2h	JP PO, nn	F2h	JP P, nn
E3h	EX (SP), HL	F3h	DI
E4h	CALL PO, nn	F4h	CALL P, nn
E5h	PUSH HL	F5h	PUSH AF
E6h	AND n	F6h	OR n
E7h	RST 20	F7h	RST 30
E8h	RET PE	F8h	RET M

Opcode	Mnemonic	Opcode	Mnemonic
E9h	JP (HL)	F9h	LD SP, HL
EAh	JP PE, nn	FAh	JP M, nn
EBh	EX DE, HL	FBh	EI
ECh	CALL PE, nn	FCh	CALL M, nn
EDh	Prefix	FDh	Prefix
EEh	XOR n	FEh	CP n
EFh	RST 28	FFh	RST 38

Table 16. Instruction set with prefix CB

Opcode	Mnemonic	Opcode	Mnemonic
00h	RLC B	10h	RL B
01h	RLC C	11h	RL C
02h	RLC D	12h	RL D
03h	RLC E	13h	RL E
04h	RLC H	14h	RL H
05h	RLC L	15h	RL L
06h	RLC (HL)	16h	RL (HL)
07h	RLC A	17h	RL A
08h	RRC B	18h	RR B
09h	RRC C	19h	RR C
0Ah	RRC D	1Ah	RR D
0Bh	RRC E	1Bh	RR E
0Ch	RRC H	1Ch	RR H
0Dh	RRC L	1Dh	RR L
0Eh	RRC (HL)	1Eh	RR (HL)
0Fh	RRC A	1Fh	RR A
20h	SLA B	30h	Not supported
21h	SLA C	31h	Not supported

Opcode	Mnemonic	Opcode	Mnemonic
22h	SLA D	32h	Not supported
23h	SLA E	33h	Not supported
24h	SLA H	34h	Not supported
25h	SLA L	35h	Not supported
26h	SLA (HL)	36h	Not supported
27h	SLA A	37h	Not supported
28h	SRA B	38h	SRL B
29h	SRA C	39h	SRL C
2Ah	SRA D	3Ah	SRL D
2Bh	SRA E	3Bh	SRL E
2Ch	SRA H	3Ch	SRL H
2Dh	SRA L	3Dh	SRL L
2Eh	SRA (HL)	3Eh	SRL (HL)
2Fh	SRA A	3Fh	SRL A
40h	BIT 0, B	50h	BIT 2, B
41h	BIT 0, C	51h	BIT 2, C
42h	BIT 0, D	52h	BIT 2, D
43h	BIT 0, E	53h	BIT 2, E
44h	BIT 0, H	54h	BIT 2, H
45h	BIT 0, L	55h	BIT 2, L
46h	BIT 0, (HL)	56h	BIT 2, (HL)
47h	BIT 0, A	57h	BIT 2, A
48h	BIT 1, B	58h	BIT 3, B
49h	BIT 1, C	59h	BIT 3, C
4Ah	BIT 1, D	5Ah	BIT 3, D
4Bh	BIT 1, E	5Bh	BIT 3, E
4Ch	BIT 1, H	5Ch	BIT 3, H
4Dh	BIT 1, L	5Dh	BIT 3, L

Opcode	Mnemonic	Opcode	Mnemonic
4Eh	BIT 1, (HL)	5Eh	BIT 3, (HL)
4Fh	BIT 1, A	5Fh	BIT 3, A
60h	BIT 4, B	70h	BIT 6, B
61h	BIT 4, C	71h	BIT 6, C
62h	BIT 4, D	72h	BIT 6, D
63h	BIT 4, E	73h	BIT 6, E
64h	BIT 4, H	74h	BIT 6, H
65h	BIT 4, L	75h	BIT 6, L
66h	BIT 4, (HL)	76h	BIT 6, (HL)
67h	BIT 4, A	77h	BIT 6, A
68h	BIT 5, B	78h	BIT 7, B
69h	BIT 5, C	79h	BIT 7, C
6Ah	BIT 5, D	7Ah	BIT 7, D
6Bh	BIT 5, E	7Bh	BIT 7, E
6Ch	BIT 5, H	7Ch	BIT 7, H
6Dh	BIT 5, L	7Dh	BIT 7, L
6Eh	BIT 5, (HL)	7Eh	BIT 7, (HL)
6Fh	BIT 5, A	7Fh	BIT 7, A
80h	RES 0, B	90h	RES 2, B
81h	RES 0, C	91h	RES 2, C
82h	RES 0, D	92h	RES 2, D
83h	RES 0, E	93h	RES 2, E
84h	RES 0, H	94h	RES 2, H
85h	RES 0, L	95h	RES 2, L
86h	RES 0, (HL)	96h	RES 2, (HL)
87h	RES 0, A	97h	RES 2, A
88h	RES 1, B	98h	RES 3, B
89h	RES 1, C	99h	RES 3, C

Opcode	Mnemonic	Opcode	Mnemonic
8Ah	RES 1, D	9Ah	RES 3, D
8Bh	RES 1, E	9Bh	RES 3, E
8Ch	RES 1, H	9Ch	RES 3, H
8Dh	RES 1, L	9Dh	RES 3, L
8Eh	RES 1, (HL)	9Eh	RES 3, (HL)
8Fh	RES 1, A	9Fh	RES 3, A
A0h	RES 4, B	B0h	RES 6, B
A1h	RES 4, C	B1h	RES 6, C
A2h	RES 4, D	B2h	RES 6, D
A3h	RES 4, E	B3h	RES 6, E
A4h	RES 4, H	B4h	RES 6, H
A5h	RES 4, L	B5h	RES 6, L
A6h	RES 4, (HL)	B6h	RES 6, (HL)
A7h	RES 4, A	B7h	RES 6, A
A8h	RES 5, B	B8h	RES 7, B
A9h	RES 5, C	B9h	RES 7, C
AAh	RES 5, D	BAh	RES 7, D
ABh	RES 5, E	BBh	RES 7, E
ACh	RES 5, H	BCh	RES 7, H
ADh	RES 5, L	BDh	RES 7, L
A Eh	RES 5, (HL)	BEh	RES 7, (HL)
AFh	RES 5, A	BFh	RES 7, A
C0h	SET 0 ,B	D0h	SET 2 ,B
C1h	SET 0, C	D1h	SET 2, C
C2h	SET 0, D	D2h	SET 2, D
C3h	SET 0, E	D3h	SET 2, E
C4h	SET 0, H	D4h	SET 2, H
C5h	SET 0, L	D5h	SET 2, L

Opcode	Mnemonic	Opcode	Mnemonic
C6h	SET 0, (HL)	D6h	SET 2, (HL)
C7h	SET 0, A	D7h	SET 2, A
C8h	SET 1, B	D8h	SET 3, B
C9h	SET 1, C	D9h	SET 3, C
CAh	SET 1, D	DAh	SET 3, D
CBh	SET 1, E	DBh	SET 3, E
CCh	SET 1, H	DCh	SET 3, H
CDh	SET 1, L	DDh	SET 3, L
CEh	SET 1, (HL)	DEh	SET 3, (HL)
CFh	SET 1, A	DFh	SET 3, A
E0h	SET 4, B	F0h	SET 6, B
E1h	SET 4, C	F1h	SET 6, C
E2h	SET 4, D	F2h	SET 6, D
E3h	SET 4, E	F3h	SET 6, E
E4h	SET 4, H	F4h	SET 6, H
E5h	SET 4, L	F5h	SET 6, L
E6h	SET 4, (HL)	F6h	SET 6, (HL)
E7h	SET 4, A	F7h	SET 6, A
E8h	SET 5, B	F8h	SET 7, B
E9h	SET 5, C	F9h	SET 7, C
EAh	SET 5, D	FAh	SET 7, D
EBh	SET 5, E	FBh	SET 7, E
ECh	SET 5, H	FCh	SET 7, H
EDh	SET 5, L	FDh	SET 7, L
EEh	SET 5, (HL)	FEh	SET 7, (HL)
EFh	SET 5, A	FFh	SET 7, A

Table 17. Instruction set with prefix DD

Opcode	Mnemonic	Opcode	Mnemonic
09h	ADD IX, BC	72h	LD (IX+d), D
19h	ADD IX, DE	73h	LD (IX+d), E
21h	LD IX, nn	74h	LD (IX+d), H
22h	LD (nn), IX	75h	LD (IX+d), L
23h	INC IX	77h	LD (IX+d), A
29h	ADD IX, XI	7Eh	LD A, (IX+d)
2Ah	LD IX, (nn)	86h	ADD A, (IX+d)
2Bh	DEC IX	8Eh	ADC A, (IX+d)
34h	INC (IX+d)	96h	SUB (IX+d)
35h	DEC (IX+d)	9Eh	SBC A, (IX+d)
36h	LD (IX+d), n	A6h	AND (IX+d)
39h	ADD IX, SP	A Eh	XOR (IX+d)
46h	LD B, (IX+d)	B6h	OR (IX+d)
4Eh	LD C, (IX+d)	B Eh	CP (IX+d)
56h	LD D, (IX+d)	E1h	POP IX
5Eh	LD E, (IX+d)	E3h	EX (SP), IX
66h	LD H, (IX+d)	E5h	PUSH IX
6Eh	LD L, (IX+d)	E9h	JP (IX)
70h	LD (IX+d), B	F9h	LD SP, IX
71h	LD (IX+d), C		

Table 18. Instruction set with prefix DD and CB

Opcode	Mnemonic	Opcode	Mnemonic
06h	RLC (IX+d)	86h	RES 0, (IX+d)
0Eh	RRC (IX+d)	8Eh	RES 1, (IX+d)
16h	RL (IX+d)	96h	RES 2, (IX+d)
1Eh	RR (IX+d)	9Eh	RES 3, (IX+d)

Opcode	Mnemonic	Opcode	Mnemonic
26h	SLA (IX+d)	A6h	RES 4, (IX+d)
2Eh	SRA (IX+d)	Aeh	RES 5, (IX+d)
3Eh	SRL (IX+d)	B6h	RES 6, (IX+d)
46h	BIT 0, (IX+d)	BEh	RES 7, (IX+d)
4Eh	BIT 1, (IX+d)	C6h	SET 0, (IX+d)
56h	BIT 2, (IX+d)	CEh	SET 1, (IX+d)
5Eh	BIT 3, (IX+d)	D6h	SET 2, (IX+d)
66h	BIT 4, (IX+d)	DEh	SET 3, (IX+d)
6Eh	BIT 5, (IX+d)	E6h	SET 4, (IX+d)
76h	BIT 6, (IX+d)	EEh	SET 5, (IX+d)
7Eh	BIT 6, (IX+d)	F6h	SET 6, (IX+d)
		FEh	SET 7, (IX+d)

Table 19. Instruction set with prefix ED

Opcode	Mnemonic	Opcode	Mnemonic
40h	IN B, (C)	62h	SBC HL, HL
41h	OUT (C), B	67h	RRD
42h	SBC HL, BC	68h	IN L, (C)
43h	LD (nn), BC	69h	OUT (C), L
44h	NEG	6Ah	ADC HL, HL
45h	RETN	6Fh	RLD
46h	IM 0	72h	SBC HL, SP
47h	LD I,A	73h	LD (nn), SP
48h	IN C, (C)	78h	IN A, (C)
49h	OUT (C), C	79h	OUT (C), A
4Ah	ADC HL, BC	7Ah	ADC HL, SP
4Bh	LD BC, (nn)	7Bh	LD SP, (nn)
4Dh	RETI	A0h	LDI
4Fh	-	A1h	CPI

Opcode	Mnemonic	Opcode	Mnemonic
50h	IN D, (C)	A2h	INI
51h	OUT (C), D	A3h	OUTI
52h	SBC HL, DE	A8h	LDD
53h	LD (nn), DE	A9h	CPD
56h	IM 1	AAh	IND
57h	LD A, I	ABh	OUTD
58h	IN E, (C)	B0h	LDIR
59h	OUT (C), E	B1h	CPIR
5Ah	ADC HL, DE	B2h	INIR
5Bh	LD DE, (nn)	B3h	OTIR
5Eh	IM 2	B8h	LDDR
5Fh	-	B9h	CPDR
60h	IN H, (C)	BAh	INDR
61h	OUT (C), H	BBh	OTDR

Table 20. Instruction set with prefix FD

Opcode	Mnemonic	Opcode	Mnemonic
09h	ADD IY, BC	72h	LD (IY+d), D
19h	ADD IY, DE	73h	LD (IY+d), E
21h	LD IY, nn	74h	LD (IY+d), H
22h	LD (nn), IY	75h	LD (IY+d), L
23h	INC IY	77h	LD (IY+d), A
29h	ADD IY, IY	7Eh	LD A, (IY+d)
2Ah	LD IY, (nn)	86h	ADD A, (IY+d)
2Bh	DEC IY	8Eh	ADC A, (IY+d)
34h	INC (IY+d)	96h	SUB (IY+d)
35h	DEC (IY+d)	9Eh	SBC A, (IY+d)
36h	LD (IY+d), n	A6h	AND (IY+d)

Opcode	Mnemonic	Opcode	Mnemonic
39h	ADD IY, SP	AEh	XOR (IY+d)
46h	LD B, (IY+d)	B6h	OR (IY+d)
4Eh	LD C, (IY+d)	BEh	CP (IY+d)
56h	LD D, (IY+d)	E1h	POP IY
5Eh	LD E, (IY+d)	E3h	EX (SP), IY
66h	LD H, (IY+d)	E5h	PUSH IY
6Eh	LD L, (IY+d)	E9h	JP (IY)
70h	LD (IY+d), B	F9h	LD SP, IY
71h	LD (IY+d), C		

Table 21. Instruction set with prefix FD and CB

Opcode	Mnemonic	Opcode	Mnemonic
06h	RLC (IY+d)	86h	RES 0, (IY+d)
0Eh	RRC (IY+d)	8Eh	RES 1, (IY+d)
16h	RL (IY+d)	96h	RES 2, (IY+d)
1Eh	RR (IY+d)	9Eh	RES 3, (IY+d)
26h	SLA (IY+d)	A6h	RES 4, (IY+d)
2Eh	SRA (IY+d)	AEh	RES 5, (IY+d)
3Eh	SRL (IY+d)	B6h	RES 6, (IY+d)
46h	BIT 0, (IY+d)	BEh	RES 7, (IY+d)
4Eh	BIT 1, (IY+d)	C6h	SET 0, (IY+d)
56h	BIT 2, (IY+d)	CEh	SET 1, (IY+d)
5Eh	BIT 3, (IY+d)	D6h	SET 2, (IY+d)
66h	BIT 4, (IY+d)	DEh	SET 3, (IY+d)
6Eh	BIT 5, (IY+d)	E6h	SET 4, (IY+d)
76h	BIT 6, (IY+d)	EEh	SET 5, (IY+d)
7Eh	BIT 6, (IY+d)	F6h	SET 6, (IY+d)
		FEh	SET 7, (IY+d)

Instruction set – detailed reference

ADC A, s

Function: Add 8-bit with carry

Description: Adds the data from the byte variable s and the Carry flag to the contents of the Accumulator and stores the result in the Accumulator. The byte variable s can be any of the following: a register r, an immediate data value n, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement d.

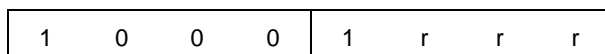
ADC A, r

Operation: ADC

$$(A) \leftarrow (A) + (r) + (C)$$

Bytes: 1

Encoding:



The table below shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

ADC A, n

Operation: ADC
 $(A) \leftarrow (A) + n + (C)$

Bytes: 2

Encoding:

1	1	0	0	1	1	1	0
n - immediate data							

ADC A, (HL)

Operation: ADC
 $(A) \leftarrow (A) + ((HL)) + (C)$

Bytes: 1

Encoding:

1	0	0	0	1	1	1	0
---	---	---	---	---	---	---	---

ADC A, (IX+d)

Operation: ADC
 $(A) \leftarrow (A) + ((IX+d)) + (C)$

Bytes: 3

Encoding:

1	1	0	1	1	1	0	1
1	0	0	0	1	1	1	0
d							

ADC A, (IY+d)

Operation: ADC
 $(A) \leftarrow (A) + ((IY+d)) + (C)$

Bytes: 3

Encoding:

1	1	1	1	1	1	0	1
1	0	0	0	1	1	1	0
d							

Flag setting (in register F):

TSK80x MCU

S	Set if result is negative; reset otherwise
Z	Set if result is zero; reset otherwise
H	Set if carry from bit 3; reset otherwise
P/V	Set if overflow; reset otherwise
N	Reset
C	Set if carry from bit 7; reset otherwise.

ADC HL, pp

Function: Add 16-bit with carry

Description: Adds the contents of register pp and the Carry flag, to the contents of the HL register. The result is stored in the HL register.

Operation: ADC
 $(HL) \leftarrow (HL) + (pp) + (C)$

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
0	1	p	p	1	0	1	0

The table below shows the registers that pp can represent. The listed 2-bit value for each register replaces the pp entry in the encoding for the instruction.

Registers	pp
BC	00
DE	01
HL	10
SP	11

Flag setting (in register F):

S	Set if result is negative; reset otherwise
Z	Set if result is zero; reset otherwise
H	Set if carry from bit 11; reset otherwise
P/V	Set if overflow; reset otherwise
N	Reset
C	Set if carry from bit 15; reset otherwise.

ADD A, s

Function: Add 8-bit data

Description: Adds the data from the byte variable s to the Accumulator and stores the result in the Accumulator. The byte variable s can be any of the following: a register r, an immediate data value n, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement d.

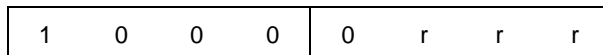
ADD A, r

Operation: ADD

$$(A) \leftarrow (A) + (r)$$

Bytes: 1

Encoding:



The table below shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

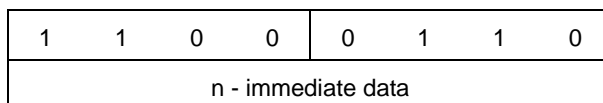
ADD A, n

Operation: ADD

$$(A) \leftarrow (A) + n$$

Bytes: 2

Encoding:



TSK80x MCU

ADD A, (HL)

Operation: ADD

$(A) \leftarrow (A) + ((HL))$

Bytes: 1

Encoding:

1	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

ADD A, (IX+d)

Operation: ADD

$(A) \leftarrow (A) + ((IX+d))$

Bytes: 3

Encoding:

1	1	0	1	1	1	0	1
1	0	0	0	0	1	1	0
d							

ADD A, (IY+d)

Operation: ADD

$(A) \leftarrow (A) + ((IY+d))$

Bytes: 3

Encoding:

1	1	1	1	1	1	0	1
1	0	0	0	0	1	1	0
d							

Flag setting (in register F):

- S Set if result is negative; reset otherwise
- Z Set if result is zero; reset otherwise
- H Set if carry from bit 3; reset otherwise
- P/V Set if overflow; reset otherwise
- N Reset
- C Set if carry from bit 7; reset otherwise.

ADD HL, pp

Function: Add 16-bit data

Description: Adds the contents of register pp to the contents of register HL and stores the result in the HL register.

Operation: ADD
 $(HL) \leftarrow (HL) + (pp)$

Bytes: 1

Encoding:

0	0	p	p	1	0	0	1
---	---	---	---	---	---	---	---

The table below shows the registers that pp can represent. The listed 2-bit value for each register replaces the pp entry in the encoding for the instruction.

Registers	pp
BC	00
DE	01
HL	10
SP	11

Flag setting (in register F):

S	Not affected
Z	not affected
H	Set if carry from bit 11; reset otherwise
P/V	not affected
N	Reset
C	Set if carry from bit 15; reset otherwise

ADD IX, pp

Function: Add 16-bit data

Description: Adds the contents of register pp to the contents of register IX and stores the result in the IX register.

Operation: ADD
 $(IX) \leftarrow (IX) + (pp)$

Bytes: 2

Encoding:

TSK80x MCU

1	1	0	1	1	1	0	1
0	0	p	p	1	0	0	1

The table below shows the registers that pp can represent. The listed 2-bit value for each register replaces the pp entry in the encoding for the instruction.

Registers	pp
BC	00
DE	01
IX	10
SP	11

Flag setting (in register F):

S	Not affected
Z	not affected
H	Set if carry from bit 11; reset otherwise
P/V	not affected
N	Reset
C	Set if carry from bit 15; reset otherwise

ADD IY, pp

Function: Add 16-bit data

Description: Adds the contents of register pp to the contents of register IY and stores the result in the IY register

Operation: ADD

$$(IY) \leftarrow (IY) + (pp)$$

Bytes: 2

Encoding:

1	1	1	1	1	1	0	1
0	0	p	p	1	0	0	1

The table below shows the registers that pp can represent. The listed 2-bit value for each register replaces the pp entry in the encoding for the instruction.

Registers	pp
BC	00
DE	01
IY	10
SP	11

Flag setting (in register F):

S	Not affected
Z	not affected
H	Set if carry from bit 11; reset otherwise
P/V	not affected
N	Reset
C	Set if carry from bit 15; reset otherwise.

AND A, s

Function: Logical AND

Description: Performs a logical AND between the data from byte variable s and the contents of the Accumulator. The byte variable s can be any of the following: a register r, an immediate data value n, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement d.

AND A, r

Operation: AND

$$(A) \leftarrow (A) \wedge (r)$$

Bytes: 1

Encoding:

1	0	1	0	0	r	r	r
---	---	---	---	---	---	---	---

TSK80x MCU

The table below shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

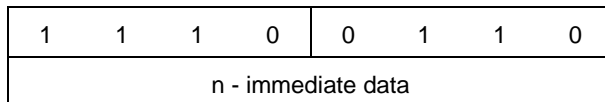
AND A, n

Operation: ADD

$$(A) \leftarrow (A) \wedge n$$

Bytes: 2

Encoding:



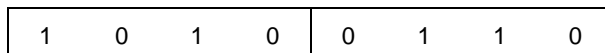
AND A, (HL)

Operation: AND

$$(A) \leftarrow (A) \wedge ((HL))$$

Bytes: 1

Encoding:



AND A, (IX+d)

Operation: AND

$$(A) \leftarrow (A) \wedge ((IX+d))$$

Bytes: 3

Encoding:

1	1	0	1	1	1	0	1
1	0	1	0	0	1	1	0
d							

AND A, (IY+d)

Operation: AND

$$(A) \leftarrow (A) \wedge ((IY+d))$$

Bytes: 3

Encoding:

1	1	1	1	1	1	0	1
1	0	1	0	0	1	1	0
d							

Flag setting (in register F):

- S Set if result is negative; reset otherwise
- Z Set if result is zero; reset otherwise
- H Set
- P/V Set if parity even; reset otherwise
- N Reset
- C Reset.

BIT b, m

Function: Test bit

Description: Tests bit b of the data from byte variable m. If it is zero the Z flag is set, otherwise the Z flag is cleared. The byte variable m can be any of the following: a register r, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement d.

The bit position within the byte of data (bit0 – bit7) is specified by using 3-bit encoding. The table below shows this encoding for each possible bit position. This 3-bit code replaces the bbb entry in the encoding for the instruction.

TSK80x MCU

Bit position	bbb
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

BIT b, r

Operation: BIT

Z flag ← NOT (bit b of (r))

Bytes: 2

Encoding:

1	1	0	0	1	0	1	1
0	1	b	b	b	r	r	r

The table below shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

BIT b, (HL)

Operation: BIT
 Z flag ← NOT (bit b of ((HL)))

Bytes: 2

Encoding:

1	1	0	0	1	0	1	1
0	1	b	b	b	1	1	0

BIT b, (IX+d)

Operation: BIT
 Z flag ← NOT (bit b of ((IX+d)))

Bytes: 4

Encoding:

1	1	0	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
0	1	b	b	b	1	1	0

BIT b, (IY+d)

Operation: BIT
 Z flag ← NOT (bit b of ((IY+d)))

Bytes: 4

Encoding:

1	1	1	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
0	1	b	b	b	1	1	0

Flag setting (in register F):

- S Set if b=7 and bit 7 in m=1
- Z Set if specified bit is zero; reset otherwise
- H Set
- P/V Set if specified bit is zero; reset otherwise
- N Reset

C Not affected.

CALL cc nn

Function: Call to subroutine if condition true

Description: Tests condition cc. If it is TRUE, the current contents of the Program Counter (PC) are pushed on to the top of the stack. The destination address nn is then loaded into the PC. This is the address of the first instruction of the subroutine. (the second n operand is the least significant byte and is loaded first)

The program then makes the jump to the subroutine, with the return address stored on the stack. Note that the return address points to the next sequential instruction after the CALL.

The PC will be incremented by three before the push is executed. The contents of the PC are pushed onto the stack as follows:

Stack Pointer (SP) contents decremented

High order byte of PC contents loaded to memory location pointed to by SP

SP decremented again

Low order byte of PC contents loaded to memory location pointed to by SP (this memory location is now at the top of the stack)

In the case when the condition is FALSE, the PC is incremented as normal and the next sequential instruction is fetched.

The table below shows the conditions that cc can represent. Each condition tests the state of a flag in the F register. The listed 3-bit value for each condition replaces the ccc entry in the encoding for the instruction.

Condition cc	Description	Relevant flag	ccc
NZ	Non Zero	Z	000
Z	Zero	Z	001
NC	Non Carry	C	010
C	Carry	C	011
PO	Parity odd	P/V	100
PE	Parity even	P/V	101
P	Sign positive	S	110
M	Sign negative	S	111

Operation: CALL

If cc true:

(SP) ← (SP) - 1

$((SP)) \leftarrow (PC_H)$
 $(SP) \leftarrow (SP) - 1$
 $((SP)) \leftarrow (PC_L)$
 $(PC) \leftarrow nn$
 Else: nothing

Bytes: 3

Encoding:

1	1	c	c	c	1	0	0
n							
n							

Flag setting (in register F): Flags are not affected.

CALL nn

Function: Call to subroutine

Description: The current contents of the Program Counter (PC) are pushed on to the top of the stack. The destination address nn is then loaded into the PC. This is the address of the first instruction of the subroutine. (The second n operand is the least significant byte and is read first)

The program then makes the jump to the subroutine, with the return address stored on the stack. Note that the return address points to the next sequential instruction after the CALL.

The PC will be incremented by three before the push is executed. The contents of the PC are pushed onto the stack as follows:

Stack Pointer (SP) contents decremented

High order byte of PC contents loaded to memory location pointed to by SP

SP decremented again

Low order byte of PC contents loaded to memory location pointed to by SP (this memory location is now at the top of the stack)

Operation: CALL

$(SP) \leftarrow (SP) - 1$

$((SP)) \leftarrow (PC_H)$

$(SP) \leftarrow (SP) - 1$

$((SP)) \leftarrow (PC_L)$

$(PC) \leftarrow nn$

Bytes: 3

TSK80x MCU

Encoding:

1	1	0	0	1	1	0	1
n							
n							

Flag setting (in register F): Flags are not affected.

CCF

Function: Complement Carry flag

Description: The Carry flag in register F (flag register) is inverted.

Operation: CCF

$(C) \leftarrow \text{not}(C)$

Bytes: 1

Encoding:

0	0	1	1	1	1	1	1
---	---	---	---	---	---	---	---

Flag setting (in register F):

S Not affected
Z Not affected
H Previous Carry flag setting
P/V Not affected
N Reset
C not (C).

CP s

Function: Compare data

Description: Compares the data stored in the byte variable *s* with that stored in the Accumulator. The contents of *s* are subtracted from the contents of the Accumulator. The contents of the Accumulator remains unchanged and the result is not stored anywhere. Only the appropriate flags in the flag register (F) are changed to reflect the result of the comparison.

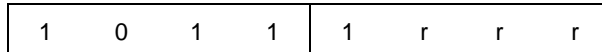
The byte variable *s* can be any of the following: a register *r*, an immediate data value *n*, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement *d*.

CP r

Operation: CP
(A) - (r)

Bytes: 1

Encoding:



The table below shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

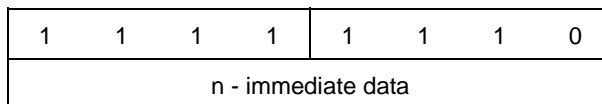
Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

CP n

Operation: CP
(A) - n

Bytes: 2

Encoding:

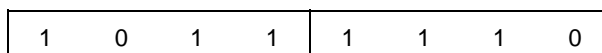


CP (HL)

Operation: CP
(A) - ((HL))

Bytes: 1

Encoding:



TSK80x MCU

CP (IX+d)

Operation: CP
(A) - ((IX+d))

Bytes: 3

Encoding:

1	1	0	1	1	1	0	1
1	0	1	1	1	1	1	0
d							

CP (IY+d)

Operation: CP
(A) - ((IY+d))

Bytes: 3

Encoding:

1	1	1	1	1	1	0	1
1	0	1	1	1	1	1	0
d							

Flag setting (in register F):

- S Set if result is negative; reset otherwise
- Z Set if result is zero; reset otherwise
- H Set if borrow from bit 4; reset otherwise
- P/V Set if overflow; reset otherwise
- N Set
- C Set if borrow; reset otherwise.

CPD

Function: Compare and decrement

Description: Compares the data stored in memory, at the location selected by the contents of the HL register, with that stored in the Accumulator. The contents of the memory location are subtracted from the contents of the Accumulator. The contents of the Accumulator remains unchanged and the result is not stored anywhere. Only the appropriate flags in the flag register (F) are changed to reflect the result of the comparison.

The HL and BC (byte counter) registers are decremented.

Operation: CPD

(A) - ((HL))
 (HL) ← (HL) - 1
 (BC) ← (BC) - 1

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
1	0	1	0	1	0	0	1

Flag setting (in register F):

S Set if result is negative; reset otherwise
 Z Set if result is zero ((A) = ((HL))); reset otherwise
 H Set if borrow from bit 4; reset otherwise
 P/V Reset if (BC) = 0; set otherwise
 N Set
 C Not affected

CPDR

Function: Compare and decrement, repeat until (BC) = 0

Description: Compares the data stored in memory, at the location selected by the contents of the HL register, with that stored in the Accumulator. The contents of the memory location are subtracted from the contents of the Accumulator. The contents of the Accumulator remains unchanged and the result is not stored anywhere. Only the appropriate flags in the flag register (F) are changed to reflect the result of the comparison.

The HL and BC (byte counter) registers are decremented. If decrementing causes BC to go to zero, or if the contents of the Accumulator and the memory location addressed by HL are equal, the instruction is terminated. Otherwise, the Program Counter is decremented by two and the instruction is repeated. Interrupt requests can be recognized.

Operation: CPDR

(A) - ((HL))
 (HL) ← (HL) - 1
 (BC) ← (BC) - 1
 If (A) - ((HL)) = 0 or (BC) = 0 then
 Finish instruction
 else
 repeat
 end

TSK80x MCU

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
1	0	1	1	1	0	0	1

Flag setting (in register F):

S	Set if result is negative; reset otherwise
Z	Set if result is zero; reset otherwise
H	Set if borrow from bit 4; reset otherwise
P/V	Reset if (BC) = 0; set otherwise
N	Set
C	Not affected

CPI

Function: Compare and increment

Description: Compares the data stored in memory, at the location selected by the contents of the HL register, with that stored in the Accumulator. The contents of the memory location are subtracted from the contents of the Accumulator. The contents of the Accumulator remains unchanged and the result is not stored anywhere. Only the appropriate flags in the flag register (F) are changed to reflect the result of the comparison.

The HL register is incremented and the BC (byte counter) register is decremented.

Operation: CPI

(A) - ((HL))

(HL) ← (HL) + 1

(BC) ← (BC) - 1

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
1	0	1	0	0	0	0	0

Flag setting (in register F):

S	Set if result is negative; reset otherwise
Z	Set if result is zero; reset otherwise
H	Set if borrow from bit 4; reset otherwise
P/V	Reset if (BC) = 0; set otherwise

N Set
 C Not affected.

CPIR

Function: Compare and increment, repeat until (BC) = 0

Description: Compares the data stored in memory, at the location selected by the contents of the HL register, with that stored in the Accumulator. The contents of the memory location are subtracted from the contents of the Accumulator. The contents of the Accumulator remains unchanged and the result is not stored anywhere. Only the appropriate flags in the flag register (F) are changed to reflect the result of the comparison.

The HL register is incremented and the BC (byte counter) register is decremented. If decrementing causes the contents of BC to go to zero, or if the contents of the Accumulator and the memory location addressed by HL are equal, the instruction is terminated. Otherwise, the Program Counter is decremented by two and the instruction is repeated. Interrupt requests can be recognized.

Operation: CPIR
 (A) - ((HL))
 (HL) ← (HL) + 1
 (BC) ← (BC) - 1
 If (A) - ((HL)) = 0 or (BC) = 0 then
 Finish instruction
 else
 repeat
 end

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
1	0	1	1	0	0	0	1

Flag setting (in register F):

S Set if result is negative; reset otherwise
 Z Set if result is zero; reset otherwise
 H Set if borrow from bit 4; reset otherwise
 P/V Reset if (BC) = 0 ; set otherwise
 N Set
 C Not affected.

CPL

Function: Complement Accumulator

Description: The contents of the Accumulator are inverted (one's complement).

Operation: CPL

$$(A) \leftarrow \text{not } (A)$$

Bytes: 1

Encoding:

0	0	1	0	1	1	1	1
---	---	---	---	---	---	---	---

Flag setting (in register F):

S Not affected

Z Not affected

H Set

P/V Not affected

N Set

C Not affected.

DAA

Function: Decimal adjust Accumulator

Description: Adjusts the contents of the Accumulator so that the result of BCD (Binary Coded Decimal) addition/subtraction operations is correctly represented in BCD format too.

Operation: DAA

$$(A) \leftarrow (A) + \text{corrective adjustment (for addition operations)}$$

$$(A) \leftarrow (A) - \text{corrective adjustment (for subtraction operations)}$$

Bytes: 1

Encoding:

0	0	1	0	0	1	1	1
---	---	---	---	---	---	---	---

The following table indicates the operation (corrective adjustment) performed for both addition operations (N=0) and subtraction operations (N=1).

N	C and H flags before DAA	Accumulator low order nibble	Accumulator high order nibble	Corrective adjustment	C flag after DAA
0	00	0-9	0-9	00	0

N	C and H flags before DAA	Accumulator low order nibble	Accumulator high order nibble	Corrective adjustment	C flag after DAA
			A-F	60	1
		A-F	0-8	06	0
			9-F	66	1
	01	0-9	0-9	06	0
			A-F	66	1
		A-F	0-9	06	0
			A-F	66	1
	10	0-9	0-F	60	1
		A-F	0-F	66	1
	11	0-F	0-F	66	1
	1	00	A-F	9-F	66
0-8				06	0
0-9			A-F	60	1
			0-9	00	0
01		0-9	A-F	66	1
			0-9	06	0
		A-F	9-F	66	1
			0-8	06	0
10		A-F	0-F	66	1
		0-9	0-F	60	1
11		0-F	0-F	66	1

Flag setting (in register F):

- S Set if MSB in Accumulator is 1 after operation; reset otherwise
- Z Set if the Accumulator is 0 after operation; reset otherwise
- H Set according to table above
- P/V Set if the Accumulator has even parity after operation; reset otherwise
- N Not affected

TSK80x MCU

C Set according to table above

DEC pp

Function: Decrement 16-bit register

Description: The contents of the 16-bit register pp are decremented.

Operation: DEC

$$(pp) \leftarrow (pp) - 1$$

Bytes: 1

Encoding:

0	0	p	p	1	0	1	1
---	---	---	---	---	---	---	---

The table below shows the registers that pp can represent. The listed 2-bit value for each register replaces the pp entry in the encoding for the instruction.

Registers	pp
BC	00
DE	01
HL	10
SP	11

Flag setting (in register F): Flags are not affected

DEC IX

Function: Decrement 16-bit register

Description: The contents of the Index register IX are decremented

Operation: DEC

$$(IX) \leftarrow (IX) - 1$$

Bytes: 2

Encoding:

1	1	0	1	1	1	0	1
0	0	1	0	1	0	1	1

Flag setting (in register F): Flags are not affected

DEC IY

Function: Decrement 16-bit register

Description: The contents of the Index register IY are decremented

Operation: DEC

$$(IY) \leftarrow (IY) - 1$$

Bytes: 2

Encoding:

1	1	1	1	1	1	0	1
0	0	1	0	1	0	1	1

Flag setting (in register F): Flags are not affected.

DEC m

Function: Decrement 8-bit data

Description: The data from the byte variable m is decremented by one and the relevant flags in the flag register (F) are set/reset accordingly. The byte variable m can be any of the following: a register r, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement d.

DEC r

Operation: DEC

$$(r) \leftarrow (r) - 1$$

Bytes: 1

Encoding:

0	0	r	r	r	1	0	1
---	---	---	---	---	---	---	---

The following table shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

TSK80x MCU

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

DEC (HL)

Operation: DEC

$$((HL)) \leftarrow ((HL)) - 1$$

Bytes: 1

Encoding:

0	0	1	1	0	1	0	1
---	---	---	---	---	---	---	---

DEC (IX+d)

Operation: DEC

$$((IX+d)) \leftarrow ((IX+d)) - 1$$

Bytes: 3

Encoding:

1	1	0	1	1	1	0	1
0	0	1	1	0	1	0	1
d							

DEC (IY+d)

Operation: DEC

$$((IY+d)) \leftarrow ((IY+d)) - 1$$

Bytes: 3

Encoding:

1	1	1	1	1	1	0	1
0	0	1	1	0	1	0	1
d							

Flag setting (in register F):

- S Set if result is negative; reset otherwise
- Z Set if result is zero; reset otherwise
- H Set if borrow from bit 3; reset otherwise
- P/V Set if data value was 80h before operation; reset otherwise
- N Set
- C Not affected.

DI

Function: Disable interrupt

Description: This instruction disables maskable interrupts by resetting the Interrupt Enable registers IFF1 and IFF2. During execution of this instruction, maskable interrupts are not serviced.

Operation: DI

$(IFF1) \leftarrow 0$

$(IFF2) \leftarrow 0$

Bytes: 1

Encoding:

1	1	1	1	0	0	1	1
---	---	---	---	---	---	---	---

Flag setting (in register F): Flags are not affected.

DJNZ e

Function: Decrement and jump if not zero

Description: This instruction decrements register B and tests its value. If it is zero, it goes to the next sequential instruction after the DJNZ instruction. If the value in register B is not zero, a displacement e is added to the contents of the Program Counter and the next instruction is fetched from the resulting address. In this case, a jump can be made in the range of -126 to 129 bytes relative to the address of the DJNZ instruction.

Note that register B is decremented first and then tested to see if its value is zero.

Operation: DJNZ

$(B) \leftarrow (B) - 1$

if $(B) = 0$ then

Go to next instruction

else

$(PC) \leftarrow (PC) + e$

Bytes: 2

TSK80x MCU

Encoding:

0	0	0	1	0	0	0	0
e							

Flag setting (in register F): Flags are not affected.

EI

Function: Enable interrupt

Description: This instruction enables maskable interrupts by setting the Interrupts Enable registers IFF1 and IFF2. During execution of this instruction and the next, maskable interrupts are not serviced.

Operation: EI

$(IFF1) \leftarrow 1$

$(IFF2) \leftarrow 1$

Bytes: 1

Encoding:

1	1	1	1	1	0	1	1
---	---	---	---	---	---	---	---

Flag setting (in register F): Flags are not affected.

EX AF, AF'

Function: Exchange contents in AF for AF'

Description: Exchange contents in register pair AF (base) with that in register pair AF' (alternative). Exchange occurs between base and alternative versions of the same register.

Operation: EX

$(A) \leftrightarrow (A')$

$(F) \leftrightarrow (F')$

Bytes: 1

Encoding:

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

Flag setting: $(F) \leftrightarrow (F')$

EX DE, HL

Function: Exchange contents in DE for HL

Description: Exchange contents in register pair DE with that in register pair HL. Exchange is carried out on a positional basis – register D with register H and register E with register L.

Operation: EX
 (D) ↔ (H)
 (E) ↔ (L)

Bytes: 1

Encoding:

1	1	1	0	1	0	1	1
---	---	---	---	---	---	---	---

Flag setting (in register F): Flags are not affected.

EX (SP), rr

Function: Exchange data on the top of the stack for that in register rr

Description: This instruction pops data from the top of the stack and stores it in register rr. The contents of register rr are pushed onto the top of the stack. The Stack Pointer register is not changed.

The variable rr can be any of the following registers: HL, IX, IY.

EX (SP), HL

Operation: EX
 (L) ↔ ((SP))
 (H) ↔ ((SP) + 1)

Bytes: 1

Encoding:

1	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---

EX (SP), IX

Operation: EX
 (IX_L) ↔ ((SP))
 (IX_H) ↔ ((SP) + 1)

Bytes: 2

TSK80x MCU

Encoding:

1	1	0	1	1	1	0	1
1	1	1	0	0	0	1	1

EX (SP), IY

Operation:

EX

$(IY_L) \leftrightarrow ((SP))$

$(IY_H) \leftrightarrow ((SP) + 1)$

Bytes:

2

Encoding:

1	1	1	1	1	1	0	1
1	1	1	0	0	0	1	1

Flag setting (in register F): Flags are not affected.

EXX

Function:

Exchange data in base registers for that in alternatives

Description:

The data in each base register (B, C, D, E, H, L) is exchanged with the value in each corresponding alternative register (B', C', D', E', H', L').

Operation:

EXX

$(B) \leftrightarrow (B')$

$(C) \leftrightarrow (C')$

$(D) \leftrightarrow (D')$

$(E) \leftrightarrow (E')$

$(H) \leftrightarrow (H')$

$(L) \leftrightarrow (L')$

Bytes:

1

Encoding:

1	1	0	1	1	0	0	1
---	---	---	---	---	---	---	---

Flag setting (in register F): Flags are not affected.

HALT

Function: Halt the microprocessor

Description: This instruction suspends CPU operation and enters into a loop in which NOP instructions are executed. The processor can leave this state only when an interrupt is received or a device reset is issued.

Operation: HALT

Bytes: 1

Encoding:

0	1	1	1	0	1	1	0
---	---	---	---	---	---	---	---

Flag setting (in register F): Flags are not affected.

IM m

Function: Set interrupt mode m

Description: This instruction sets the microcontroller's maskable interrupt mode, which defines how the processor handles an interrupt from a peripheral device. The following three interrupt modes are available:

Mode 0 - the peripheral device requesting an interrupt can place any instruction on the data bus that it requires to be executed by the TSK80A. For single byte instructions, the processor reads the instruction during acknowledgement of the interrupt request. For instructions that are more than one byte in length, the first byte is read during the interrupt acknowledgement, with the remaining byte(s) read in accordance with the processor's normal data memory read cycle.

Mode 1 - the processor ignores all data on the data bus and issues a restart - jumping directly to address 0038h. This behavior is similar to execution of the instruction RST 38h

Mode 2 - the peripheral device requesting the interrupt supplies an 8-bit vector. This data is loaded as the low order byte of the address bus, with the high order byte loaded from the contents of the TSK80A's Interrupt register (I). These two bytes point to the starting address of the first instruction in the interrupt service routine.

Operation: IM

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
0	1	0	m	m	1	1	0

The table below shows the interrupt modes that m can represent. The listed 2-bit value for each mode replaces the mm entry in the encoding for the instruction.

Mode m	mm
0	00
1	10
2	11

Flag setting (in register F): Flags are not affected.

IN A, (n)

Function: Input from port n

Description: The 8-bit immediate data value n is placed on the low order byte of the address bus. This value is used to select the I/O port (within the lowest 256 bytes of I/O address space) to which an external I/O device is connected. The contents of the Accumulator are placed on the high order byte of the address bus.

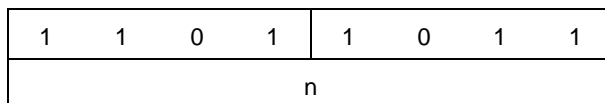
A single byte of data is read from the selected I/O port, on to the data bus, and loaded into the Accumulator.

Operation: IN

$(A) \leftarrow I/O((n))$

Bytes: 2

Encoding:



Flag setting (in register F): Flags are not affected.

IN r, (C)

Function: Input from port (C)

Description: The contents of register pair BC are placed on the address bus, (C onto the low order byte and B onto the top order byte). The low order byte of the address (contents of C) is used to select the I/O port (within the lowest 256 bytes of I/O address space) to which an external I/O device is connected.

A single byte of data is read from the selected I/O port, on to the data bus, and loaded into the register r.

Operation: IN

$(r) \leftarrow I/O((C))$

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
0	1	r	r	r	0	0	0

The table below shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Flag setting (in register F):

- S Set if bit 7 of the input data is 1; reset otherwise
- Z Set if all 8 bits of the input data are 0; reset otherwise
- H Reset
- P/V Set if the parity of the input data is even; reset otherwise
- N Reset
- C Not affected.

INC pp

Function: Increment 16-bit register

Description: The contents of the 16-bit register pp are incremented.

Operation: INC

$$(pp) \leftarrow (pp) + 1$$

Bytes: 1

Encoding:

0	0	p	p	0	0	1	1
---	---	---	---	---	---	---	---

The table below shows the registers that pp can represent. The listed 2-bit value for each register replaces the pp entry in the encoding for the instruction.

Registers	pp
BC	00
DE	01
HL	10
SP	11

Flag setting (in register F): Flags are not affected

INC IX

Function: Increment 16-bit register

Description: The contents of the Index register IX are incremented

Operation: INC

$$(IX) \leftarrow (IX) + 1$$

Bytes: 2

Encoding:

1	1	0	1	1	1	0	1
0	0	1	0	0	0	1	1

Flag setting (in register F): Flags are not affected

INC IY

Function: Increment 16-bit register

Description: The contents of the Index register IY are incremented

Operation: INC

$$(IY) \leftarrow (IY) + 1$$

Bytes: 2

Encoding:

1	1	1	1	1	1	0	1
0	0	1	0	0	0	1	1

Flag setting (in register F): Flags are not affected.

INC m

Function: Increments 8-bit data

Description: The data from the byte variable *m* is incremented by one and the relevant flags in the flag register (F) are set/reset accordingly. The byte variable *m* can be any of the following: a register *r*, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement *d*.

INC r

Operation: INC

$$(r) \leftarrow (r) + 1$$

Bytes: 1

Encoding:

0	0	r	r	r	1	0	0
---	---	---	---	---	---	---	---

The table below shows the registers that *r* can represent. The listed 3-bit value for each register replaces the *rrr* entry in the encoding for the instruction.

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

INC (HL)

Operation: INC

$$((HL)) \leftarrow ((HL)) + 1$$

Bytes: 1

Encoding:

0	0	1	1	0	1	0	0
---	---	---	---	---	---	---	---

TSK80x MCU

INC (IX+d)

Operation: INC

$$((IX+d)) \leftarrow ((IX+d)) + 1$$

Bytes: 3

Encoding:

1	1	0	1	1	1	0	1
0	0	1	1	0	1	0	0
d							

INC (IY+d)

Operation: INC

$$((IY+d)) \leftarrow ((IY+d)) + 1$$

Bytes: 3

Encoding:

1	1	1	1	1	1	0	1
0	0	1	1	0	1	0	0
d							

Flag setting (in register F):

S	Set if result is negative; reset otherwise
Z	Set if result is zero; reset otherwise
H	Set if carry from bit 3; reset otherwise
P/V	Set if data value was 7Fh before operation; reset otherwise
N	Reset
C	Not affected.

IND

Function: Input and decrement

Description: The contents of register pair BC are placed on the address bus, (C onto the low order byte and B onto the top order byte). The low order byte of the address (contents of C) is used to select the I/O port (within the lowest 256 bytes of I/O address space) to which an external I/O device is connected.

One byte of data is read from the I/O port, onto the data bus and then placed into memory at the location specified by the contents of the HL register. Both the HL and B registers are then decremented. Register B is used as a byte counter.

Operation: IND

$$((HL)) \leftarrow I/O((C))$$

$$(HL) \leftarrow (HL) - 1$$

$$(B) \leftarrow (B) - 1$$

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
1	0	1	0	1	0	1	0

Flag setting (in register F):

S Not Affected
 Z Set if (B) - 1 = 0; reset otherwise
 H Not Affected
 P/V Not affected
 N Set to 1
 C Not Affected.

INDR

Function: Input, decrement and repeat

Description: The contents of register pair BC are placed on the address bus, (C onto the low order byte and B onto the top order byte). The low order byte of the address (contents of C) is used to select the I/O port (within the lowest 256 bytes of I/O address space) to which an external I/O device is connected.

One byte of data is read from the I/O port, onto the data bus and then placed into memory at the location specified by the contents of the HL register. Both the HL and B registers are then decremented. Register B is used as a byte counter.

The content of register B is then tested. If decrementing causes the contents of B to go to zero, the next sequential instruction is fetched, as normal. Otherwise, the Program Counter is decremented by two and the INDR instruction is repeated. Interrupt requests can be recognized.

Operation: INDR

$$((HL)) \leftarrow I/O((C))$$

$$(HL) \leftarrow (HL) - 1$$

$$(B) \leftarrow (B) - 1$$

if (B) = 0 then

Next instruction

else

Repeat instruction

TSK80x MCU

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
1	0	1	1	1	0	1	0

Flag setting (in register F):

S Not Affected
Z Set if (B) - 1 = 0; reset otherwise
H Not Affected
P/V Not affected
N Set to 1
C Not Affected.

INI

Function: Input and increment

Description: The contents of register pair BC are placed on the address bus, (C onto the low order byte and B onto the top order byte). The low order byte of the address (contents of C) is used to select the I/O port (within the lowest 256 bytes of I/O address space) to which an external I/O device is connected.

One byte of data is read from the I/O port, onto the data bus and then placed into memory at the location specified by the contents of the HL register. The HL register is then incremented and register B (used as a byte counter) is decremented.

Operation: INI

$((HL)) \leftarrow I/O((C))$

$(HL) \leftarrow (HL) + 1$

$(B) \leftarrow (B) - 1$

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
1	0	1	0	0	0	1	0

Flag setting (in register F):

S Not Affected
Z Set if (B) - 1 = 0; reset otherwise
F5 Bit 5 from result (B) - 1
H Not Affected

F3 Bit 3 from result (B) - 1
 P/V Not affected
 N Set to 1
 C Not Affected.

INIR

Function: Input, increment and repeat

Description: The contents of register pair BC are placed on the address bus, (C onto the low order byte and B onto the top order byte). The low order byte of the address (contents of C) is used to select the I/O port (within the lowest 256 bytes of I/O address space) to which an external I/O device is connected.

One byte of data is read from the I/O port, onto the data bus and then placed into memory at the location specified by the contents of the HL register. The HL register is then incremented and register B (used as a byte counter) is decremented.

The content of register B is then tested. If decrementing causes the contents of B to go to zero, the next sequential instruction is fetched, as normal. Otherwise, the Program Counter is decremented by two and the INIR instruction is repeated. Interrupt requests can be recognized.

Operation: INIR
 $((HL)) \leftarrow I/O((C))$
 $(HL) \leftarrow (HL) + 1$
 $(B) \leftarrow (B) - 1$
 if (B) = 0 then
 Next instruction
 else
 Repeat instruction

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
1	0	1	1	0	0	1	0

Flag setting (in register F):

S Not Affected
 Z Set if (B) - 1 = 0; reset otherwise
 H Not Affected
 P/V Not affected
 N Set to 1

C Not Affected.

JP (rr)

Function: Jump to address in register rr

Description: The content of the 16-bit register rr is loaded into the Program Counter (PC) and the next instruction is fetched from this location. The variable rr can be one of the following 16-bit registers: HL, IX or IY.

JP (HL)

Operation: JP
(PC) ← (HL)

Bytes: 1

Encoding:

1	1	1	0	1	0	0	1
---	---	---	---	---	---	---	---

JP (IX)

Operation: JP
(PC) ← (IX)

Bytes: 2

Encoding:

1	1	0	1	1	1	0	1
1	1	1	0	1	0	0	1

JP (IY)

Operation: JP
(PC) ← (IY)

Bytes: 2

Encoding:

1	1	1	1	1	1	0	1
1	1	1	0	1	0	0	1

Flag setting (in register F): Flags are not affected.

JP cc nn

Function: Jump to address if condition is true.

Description: Tests condition cc. If it is TRUE, the destination address nn is loaded into the Program Counter (PC) and the processor fetches the next instruction from this

address. If the condition is FALSE, the PC is incremented as normal and the next sequential instruction is fetched.

The table below shows the conditions that cc can represent. Each condition tests the state of a flag in the F register. The listed 3-bit value for each condition replaces the ccc entry in the encoding for the instruction.

Condition cc	Describe	Relevant flag	ccc
NZ	Non zero	Z	000
Z	Zero	Z	001
NC	Non Carry	C	010
C	Carry	C	011
PO	Parity odd	P/V	100
PE	Parity even	P/V	101
P	Sign positive	S	110
M	Sign negative	S	111

Operation: JP
 If cc true: (PC) ← nn
 Else continue

Bytes: 3

Encoding:

1	1	c	c	c	0	1	0
n							
n							

Flag setting (in register F): Flags are not affected.

JR cc e

Function: Jump relative if condition true

Description: Tests condition cc. If it is TRUE, a displacement e is added to the contents of the Program Counter and the next instruction is fetched from the resulting address. In this case, a jump can be made in the range of -126 to 129 bytes relative to the address of the JR instruction.

If the condition is FALSE, the PC is incremented as normal and the next sequential instruction is fetched.

The table below shows the conditions that cc can represent (testing the C and Z flags in the F register). The listed 2-bit value for each condition replaces the cc entry in the encoding for the instruction.

Condition cc	Name	Relevant Flag	cc
NZ	Non Zero	Z	00
Z	Zero	Z	01
NC	Non Carry	C	10
C	Carry	C	11

Operation: JR
 If cc is true then
 $(PC) \leftarrow (PC) + e$
 Else
 continue

Bytes: 2

Encoding:

0	0	1	c	c	0	0	0
e							

Flag setting (in register F): Flags are not affected.

JP nn

Function: Jump to address

Description: The destination address nn is loaded into the Program Counter (PC) and the processor fetches the next instruction from this address.

Operation: JP
 $(PC) \leftarrow nn$

Bytes: 3

Encoding:

1	1	0	0	0	0	1	1
n							
n							

Flag setting (in register F): Flags are not affected.

JR e

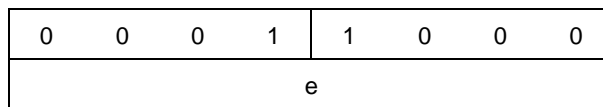
Function: Jump relative

Description: A displacement e is added to the contents of the Program Counter (PC) and the next instruction is fetched from the resulting address. In this case, a jump can be made in the range -126 to 129 bytes relative to the address of the JR instruction.

Operation: JR
 $(PC) \leftarrow (PC) + e$

Bytes: 2

Encoding:



Flag setting (in register F) : Flags are not affected

LD (aa), A

Function: Load Accumulator to memory

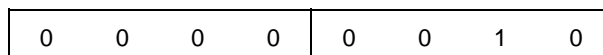
Description: The contents of the Accumulator are loaded into the memory location selected by the contents of the 16-bit variable aa. The variable aa is used to represent the following: registers BC, DE, HL, a direct address nn or the sum of an Index register (IX or IY) and an 8-bit displacement d.

LD (BC), A

Operation: LD
 $((BC)) \leftarrow (A)$

Bytes: 1

Encoding:

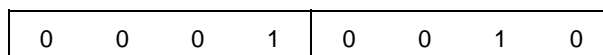


LD (DE), A

Operation: LD
 $((DE)) \leftarrow (A)$

Bytes: 1

Encoding:



TSK80x MCU

LD (HL), A

Operation: LD
((HL)) ← (A)

Bytes: 1

Encoding:

0	1	1	1	0	1	1	1
---	---	---	---	---	---	---	---

LD (nn), A

Operation: LD
(nn) ← (A)

Bytes: 3

Encoding:

0	0	1	1	0	0	1	0
n							
n							

LD (IX+d), A

Operation: LD
((IX+d)) ← (A)

Bytes: 3

Encoding:

1	1	0	1	1	1	0	1
0	1	1	1	0	1	1	1
d							

LD (IY+d), A

Operation: LD
((IY+d)) ← (A)

Bytes: 3

Encoding:

1	1	1	1	1	1	0	1
0	1	1	1	0	1	1	1
d							

Flag setting (in register F): Flags are not affected.

LD (nn), pp

Function: Load register contents to memory location

Description: The contents of the register variable pp are loaded into the memory location selected by direct address nn. The register variable pp can be any of the following: BC, DE, HL, SP, IX, or IY.

The low order byte of each register (C, E, L, SP_L, IX_L, IY_L) is loaded into memory location nn and the high order byte (B, D, H, SP_H, IX_H, IY_H) is loaded into memory location nn + 1.

LD (mn), HL

Operation: LD

(nn) ← (L)

(nn + 1) ← (H)

Bytes: 3

Encoding:

0	0	1	0	0	0	1	0
n							
n							

LD (nn), pp

Operation: LD

(nn) ← (pp_L)

(nn + 1) ← (pp_H)

Bytes: 4

Encoding:

1	1	1	0	1	1	0	1
0	1	p	p	0	0	1	1
n							
n							

The table below shows the registers that pp can represent. The listed 2-bit value for each register replaces the pp entry in the encoding for the instruction.

TSK80x MCU

Register	pp
BC	00
DE	01
HL	10
SP	11

LD (nn), IX

Operation LD
 $(nn) \leftarrow (IX_L)$
 $(nn + 1) \leftarrow (IX_H)$

Bytes 4

Encoding:

1	1	0	1	1	1	0	1
0	0	1	0	0	0	1	0
n							
n							

LD (nn), IY

Operation LD
 $(nn) \leftarrow (IY_L)$
 $(nn + 1) \leftarrow (IY_H)$

Bytes 4

Encoding:

1	1	1	1	1	1	0	1
0	0	1	0	0	0	1	0
n							
n							

Flag setting (in register F): Flags are not affected.

LD A, (aa)

Function: Load Accumulator from memory

Description: The byte data from memory is loaded into the Accumulator. The location is selected by the contents of the 16-bit register variable aa. The variable aa is used to represent

the following: registers BC, DE, HL, a direct address nn or the sum of an Index register (IX or IY) and an 8-bit displacement d.

LD A, (BC)

Operation: LD
 $(A) \leftarrow ((BC))$

Bytes: 1

Encoding:

0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---

LD A, (DE)

Operation: LD
 $(A) \leftarrow ((DE))$

Bytes: 1

Encoding:

0	0	0	1	1	0	1	0
---	---	---	---	---	---	---	---

LD A, (HL)

Operation: LD
 $(A) \leftarrow ((HL))$

Bytes: 1

Encoding:

0	1	1	1	1	1	1	0
---	---	---	---	---	---	---	---

LD A, (nn)

Operation: LD
 $(A) \leftarrow (nn)$

Bytes: 3

Encoding:

0	0	1	1	1	0	1	0
n							
n							

TSK80x MCU

LD A, (IX+d)

Operation LD
(A) ← ((IX+d))

Bytes 3

Encoding:

1	1	0	1	1	1	0	1
0	1	1	1	1	1	1	0
d							

LD A, (IY+d)

Operation LD
(A) ← ((IY+d))

Bytes 3

Encoding:

1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	0
d							

Flag setting (in register F): Flags are not affected.

LD A, I

Function: Load interrupt vector to Accumulator

Description: The contents of the Interrupt register (I) are loaded into the Accumulator. In addition, the state of flip-flop IFF2 (Interrupt Enable register for maskable interrupts) is stored in the flag register (F) using the P/V flag.

Operation: LD
(A) ← (I)

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
0	1	0	1	0	1	1	1

Flag setting (in register F):

- S Set if content of register I is negative; reset otherwise
- Z Set if content of register I is zero; reset otherwise

H Reset
 P/V Contains contents of IFF2
 N Reset
 C Not affected.

LD ee, nn

Function: Load 16-bit “immediate” data into register

Description: The 16-bit “immediate” data is loaded into the 16-bit register represented by register variable ee. The register variable ee can be any of the following: BC, DE, HL, SP, IX, IY.

The first n operand after the Opcode is the low order byte of the “immediate” data and is loaded into the low order byte of the destination register. The second n is the high order byte of the “immediate” data and is loaded into the high order byte of the destination register.

LD pp, nn

Operation: LD
 (pp_L) ← n
 (pp_H) ← n

Bytes: 3

Encoding:

0	0	s	s	0	0	0	1
n							
n							

The table below shows the registers that pp can represent. The listed 2-bit value for each register replaces the pp entry in the encoding for the instruction.

Registers	pp
BC	00
DE	01
HL	10
SP	11

TSK80x MCU

LD IX, nn

Operation LD
(IX_L) ← n
(IX_H) ← n

Bytes 3

Encoding:

1	1	0	1	1	1	0	1
0	0	1	0	0	0	0	1
n							
n							

LD IY, nn

Operation LD
(IY_L) ← n
(IY_H) ← n

Bytes 3

Encoding:

1	1	1	1	1	1	0	1
0	0	1	0	0	0	0	1
n							
n							

Flag setting (in register F): Flags are not affected.

LD ee, (nn)

Function: Load contents of memory location to register

Description: The contents of the memory location selected by direct address nn are loaded into the register variable ee. The register variable ee can be any of the following: BC, DE, HL, SP, IX, or IY.

The low order byte of each register (C, E, L, SP_L, IX_L, IY_L) is loaded with the contents of memory location nn and the high order byte (B, D, H, SP_H, IX_H, IY_H) is loaded with the contents of memory location nn + 1. In the destination address nn, the first n operand is the low order byte.

LD HL, (nn)

Operation: LD
 (L) ← (nn)
 (H) ← (nn+1)

Bytes: 3

Encoding:

0	0	1	0	1	0	1	0
n							
n							

LD IX, (nn)

Operation LD
 (IX_L) ← (nn)
 (IX_H) ← (nn+1)

Bytes 4

Encoding:

1	1	0	1	1	1	0	1
0	0	1	0	1	0	1	0
n							
n							

LD IY, (nn)

Operation LD
 (IY_L) ← (nn)
 (IY_H) ← (nn+1)

Bytes 4

Encoding:

1	1	1	1	1	1	0	1
0	0	1	0	1	0	1	0
n							
n							

TSK80x MCU

LD pp, (nn)

Operation: LD

$(pp_L) \leftarrow (nn)$

$(pp_H) \leftarrow (nn+1)$

Bytes: 3

Encoding:

1	1	1	0	1	1	0	1
0	1	p	p	1	0	1	1
n							
n							

The table below shows the registers that pp can represent. The listed 2-bit value for each register replaces the pp entry in the encoding for the instruction.

Registers	pp
BC	00
DE	01
HL	10
SP	11

Flag setting (in register F): Flags are not affected

LDI, A

Function: Load Accumulator to interrupt vector

Description: The contents of the Accumulator are loaded into the Interrupt register (I).

Operation: LD

$(I) \leftarrow (A)$

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
0	1	0	0	0	1	1	1

Flag setting (in register F): Flags are not affected.

LD m, n

Function: Load 8-bit “immediate” data

Description: The 8-bit “immediate” data is loaded into the byte variable m. The byte variable m can be any of the following: a register r, a memory location selected by the contents of the HL register, or a memory location selected by the sum of an Index register (IX or IY) and an 8-bit displacement d.

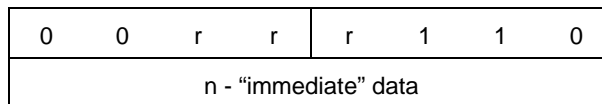
LD r, n

Operation: LD

$$(r) \leftarrow n$$

Bytes: 2

Encoding:



The table below shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

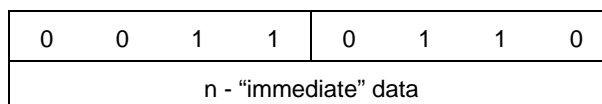
LD (HL), n

Operation: LD

$$((HL)) \leftarrow n$$

Bytes: 2

Encoding:



TSK80x MCU

LD (IX+d), n

Operation: LD
((IX+d)) ← n

Bytes: 4

Encoding:

1	1	0	1	1	1	0	1
0	0	1	1	0	1	1	0
d							
n - "immediate" data							

LD (IY+d), n

((IY+d)) ← n

Bytes: 4

Encoding:

1	1	1	1	1	1	0	1
0	0	1	1	0	1	1	0
d							
n - "immediate" data							

Flag setting (in register F): Flags are not affected.

LD m, r

Function: Load contents from register

Description: The content of register r is loaded into the byte variable m. The byte variable m can be any of the following: a register r, data from a memory location selected by the contents of the HL register, or data from a memory location selected by the sum of an Index register (IX or IY) and an 8-bit displacement.

The table below shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the relevant instruction. In the case where the contents of a register r are being loaded into a register r (e.g. (B) ← (C)), the registers are distinguished by the suffixes 1 and 2 and the encoding entries will become r1r1r1 and r2r2r2.

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

LD r1, r2

Operation: LD
 $(r1) \leftarrow (r2)$

Bytes: 1

Encoding:

0	1	r1	r1	r1	r2	r2	r2
---	---	----	----	----	----	----	----

LD (HL), r

Operation: LD
 $((HL)) \leftarrow (r)$

Bytes: 1

Encoding:

0	1	1	1	0	r	r	r
---	---	---	---	---	---	---	---

LD (IX+d), r

Operation: LD
 $((IX+d)) \leftarrow (r)$

Bytes: 3

Encoding:

1	1	0	1	1	1	0	1
0	1	1	1	0	r	r	r
d							

LD (IY+d), r

$((IY+d)) \leftarrow (r)$

Bytes: 3

Encoding:

1	1	1	1	1	1	0	1
0	1	1	1	0	r	r	r
d							

Flag setting (in register F): Flags are not affected.

LD r, m

Function: Load data into register

Description: The data from byte variable m is loaded into the register r. The byte variable m can be any of the following: a register r, 8-bit “immediate data, data from a memory location selected by the contents of the HL register, or data from a memory location selected by the sum of an Index register (IX or IY) and an 8-bit displacement.

The table below shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the relevant instruction. In the case where the contents of a register r are being loaded into a register r (e.g. (B) ← (C)), the registers are distinguished by the suffixes 1 and 2 and the encoding entries will become r1r1r1 and r2r2r2.

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

LD r1, r2

Operation: LD
 $(r1) \leftarrow (r2)$

Bytes: 1

Encoding:

0	1	r1	r1	r1	r2	r2	r2
---	---	----	----	----	----	----	----

Note: The LD H, H opcode is reserved and cannot be used. It is used to represent a software breakpoint.

LD r, n

Operation: LD
 $(r) \leftarrow n$

Bytes: 2

Encoding:

0	0	r	r	r	1	1	0
n - "immediate" data							

LD r, (HL)

Operation: LD
 $(r) \leftarrow ((HL))$

Bytes: 1

Encoding:

0	1	r	r	r	1	1	0
---	---	---	---	---	---	---	---

LD r, (IX+d)

Operation: LD
 $(r) \leftarrow ((IX+d))$

Bytes: 3

Encoding:

1	1	0	1	1	1	0	1
0	1	r	r	r	1	1	0
d							

LD r, (IY+d)

$(r) \leftarrow ((IY+d))$

Bytes: 3

Encoding:

1	1	1	1	1	1	0	1
0	1	r	r	r	1	1	0
d							

Flag setting (in register F): Flags are not affected.

LD SP, rr

Function: Load contents of register into Stack Pointer

Description: The contents of the 16-bit register variable rr are loaded into the Stack Pointer (SP). The low and high order bytes of the source register are loaded into the low and high order bytes of the Stack Pointer, respectively.

The register variable rr can be any of the following registers: HL, IX, IY.

LD SP, HL

Operation: LD

$(SP_L) \leftarrow (L)$

$(SP_H) \leftarrow (H)$

Bytes: 1

Encoding:

1	1	1	1	1	0	0	1
---	---	---	---	---	---	---	---

LD SP, IX

Operation: LD

$(SP_L) \leftarrow (IX_L)$

$(SP_H) \leftarrow (IX_H)$

Bytes: 2

Encoding:

1	1	0	1	1	1	0	1
1	1	1	1	1	0	0	1

LD SP, IY

Operation: LD
 $(SP_L) \leftarrow (IY_L)$
 $(SP_H) \leftarrow (IY_H)$

Bytes: 2

Encoding:

1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	0	0	1

LDD

Function: Load and decrement

Description: The byte of data from the memory location addressed by the contents of the register HL are loaded into the memory location addressed by the contents of the DE register. At the end of the instruction, the contents of the HL, DE and BC (byte counter) registers are decremented.

Operation: LDD
 $((DE)) \leftarrow ((HL))$
 $(HL) \leftarrow (HL) - 1$
 $(DE) \leftarrow (DE) - 1$
 $(BC) \leftarrow (BC) - 1$

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
1	0	1	0	1	0	0	0

Flag setting (in register F):

S Not affected
 Z Not affected
 H Reset
 P/V Reset if (BC) = 0; set otherwise
 N Reset
 C Not affected.

LDDR

Function: Load and decrement, repeat until (BC) = 0

Description: The byte of data from the memory location addressed by the contents of the register HL are loaded into the memory location addressed by the contents of the DE register. The contents of the HL, DE and BC (byte counter) registers are then decremented.

At the end of the instruction, the content of register BC is tested. If decrementing causes the contents of BC to go to zero, the next sequential instruction is fetched, as normal. Otherwise, the Program Counter is decremented by two and the LDDR instruction is repeated. Interrupt requests can be recognized.

Operation: LDDR
 $((DE)) \leftarrow ((HL))$
 $(HL) \leftarrow (HL) - 1$
 $(DE) \leftarrow (DE) - 1$
 $(BC) \leftarrow (BC) - 1$
 if (BC) = 0 then
 next instruction
 else
 repeat instruction

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
1	0	1	1	1	0	0	0

Flag setting (in register F):

S Not affected
 Z Not affected
 H Reset
 P/V Reset
 N Reset
 C Not affected

LDI

Function: Load and increment

Description: The byte of data from the memory location addressed by the contents of the register HL are loaded into the memory location addressed by the contents of the DE register. At the end of the instruction, the contents of the HL and DE registers are incremented. The contents of register BC (byte counter) are decremented.

Operation: LDI
 $((DE)) \leftarrow ((HL))$
 $(HL) \leftarrow (HL) + 1$
 $(DE) \leftarrow (DE) + 1$
 $(BC) \leftarrow (BC) - 1$

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
1	0	1	0	0	0	0	0

Flag setting (in register F):

S Not affected
 Z Not affected
 H Reset
 P/V Reset if (BC) = 0; set otherwise
 N Reset
 C Not affected.

LDIR

Function: Load and increment, repeat until (BC) = 0

Description: The byte of data from the memory location addressed by the contents of the register HL are loaded into the memory location addressed by the contents of the DE register. The contents of the HL and DE registers are then incremented. The contents of the BC register (byte counter) are then decremented.

At the end of the instruction, the content of register BC is tested. If decrementing causes the contents of BC to go to zero, the next sequential instruction is fetched, as normal. Otherwise, the Program Counter is decremented by two and the LDIR instruction is repeated. Interrupt requests can be recognized.

Operation: LDIR
 $((DE)) \leftarrow ((HL))$
 $(HL) \leftarrow (HL) + 1$
 $(DE) \leftarrow (DE) + 1$

TSK80x MCU

$(BC) \leftarrow (BC) - 1$
if $(BC) = 0$ then
next instruction
else
repeat instruction

Bytes:

2

Encoding:

1	1	1	0	1	1	0	1
1	0	1	1	0	0	0	0

Flag setting (in register F):

S Not affected
Z Not affected
H Reset
P/V Reset
N Reset
C Not affected.

NEG

Function: Negate Accumulator

Description: The contents of the Accumulator are subtracted from zero (negated) and the result stored in the Accumulator. Note that if the Accumulator contains the value 80h, there is no change, as the result of the operation is 80h.

Operation: NEG

$(A) \leftarrow 0 - (A)$

Bytes:

2

Encoding:

1	1	1	0	1	1	0	1
0	1	0	0	0	1	0	0

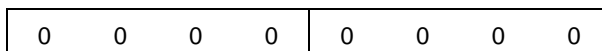
Flag setting (in register F):

S Set if result is negative; reset otherwise
Z Set if result is zero; reset otherwise
H Set if borrow from bit 4; reset otherwise

P/V Set if Accumulator was 80h before operation; reset otherwise
 N Set
 C Reset if Accumulator was 00h before operation; set otherwise.

NOP

Function: No operation
 Description: The processor does not perform an operation during this instruction cycle.
 Operation: NOP
 Bytes: 1
 Encoding:



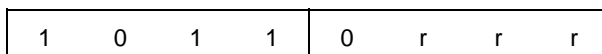
Flag setting (in register F): Flags are not affected.

OR A, s

Function: Logical OR
 Description: Performs a logical OR between the data from byte variable s and the contents of the Accumulator. The byte variable s can be any of the following: a register r, an immediate data value n, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement d.

OR A, r

Operation: OR
 $(A) \leftarrow (A) \vee (r)$
 Bytes: 1
 Encoding:



The table below shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

Register	rrr
B	000
C	001
D	010

TSK80x MCU

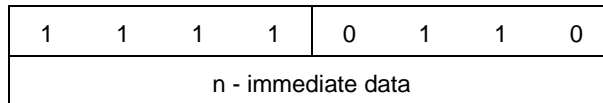
Register	rrr
E	011
H	100
L	101
A	111

OR A, n

Operation: OR
 $(A) \leftarrow (A) \vee n$

Bytes: 2

Encoding:

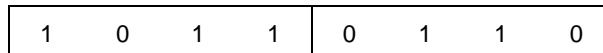


OR A, (HL)

Operation: OR
 $(A) \leftarrow (A) \vee ((HL))$

Bytes: 1

Encoding:

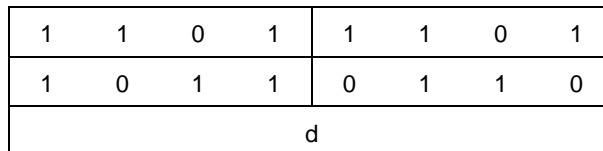


OR A, (IX+d)

Operation: OR
 $(A) \leftarrow (A) \vee ((IX+d))$

Bytes: 3

Encoding:



OR A, (IY+d)

Operation: OR
 $(A) \leftarrow (A) \vee ((IY+d))$

Bytes: 3

Encoding:

1	1	1	1	1	1	0	1
1	0	1	1	0	1	1	0
d							

Flag setting (in register F):

- S Set if result is negative; reset otherwise
- Z Set if result is zero; reset otherwise
- H Reset
- P/V Set if parity even; reset otherwise
- N Reset
- C Reset.

OTDR

Function: Output, decrement and repeat

Description: The byte of data at the memory location specified by the contents of register HL is read and temporarily stored in the CPU. Register B (byte counter) is then decremented and its contents placed on to the high order byte of the address bus. At the same time, the contents of register C are placed onto the low order byte of the address bus. The low order byte of the address (contents of C) is used to select the I/O port (within the lowest 256 bytes of I/O address space) to which an external I/O device is connected.

The temporarily stored byte of data is then placed on the data bus and written, through the selected port, to the peripheral device. The HL register is then decremented.

The content of register B is then tested. If (B) = 0, the next sequential instruction is fetched, as normal. Otherwise, the Program Counter is decremented by two and the OTDR instruction is repeated. Interrupt requests can be recognized.

Operation: OTDR
 $I/O((C)) \leftarrow ((HL))$
 $(B) \leftarrow (B) - 1$
 $(HL) \leftarrow (HL) - 1$
 if (B) = 0 then
 Next instruction
 else
 Repeat instruction

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
1	0	1	1	1	0	1	1

Flag setting (in register F):

- S Not Affected
- Z Set if (B) - 1 is zero; reset otherwise
- H: Not Affected
- P/V Not Affected
- N Set to 1
- C Not affected.

OTIR

Function: Output, increment and repeat

Description: The byte of data at the memory location specified by the contents of register HL is read and temporarily stored in the CPU. Register B (byte counter) is then decremented and its contents placed on to the high order byte of the address bus. At the same time, the contents of register C are placed onto the low order byte of the address bus. The low order byte of the address (contents of C) is used to select the I/O port (within the lowest 256 bytes of I/O address space) to which an external I/O device is connected.

The temporarily stored byte of data is then placed on the data bus and written, through the selected port, to the peripheral device. The HL register is then incremented

The content of register B is then tested. If (B) = 0, the next sequential instruction is fetched, as normal. Otherwise, the Program Counter is decremented by two and the OTDR instruction is repeated. Interrupt requests can be recognized.

Operation: OTIR
 $I/O((C)) \leftarrow ((HL))$
 $(B) \leftarrow (B) - 1$
 $(HL) \leftarrow (HL) + 1$
 if (B) = 0 then
 Next instruction
 else
 Repeat instruction

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

1	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

Flag setting (in register F):

- S Not Affected
- Z Set if (B) - 1 is zero; reset otherwise
- H Not Affected
- P/V Not Affected
- N Set to 1
- C Not affected.

OUT (C), r

Function: Output to port (C)

Description: The contents of register pair BC are placed on the address bus, (C onto the low order byte and B onto the top order byte). The low order byte of the address (contents of C) is used to select the I/O port (within the lowest 256 bytes of I/O address space) to which an external I/O device is connected.

A single byte of data is read from register r, on to the data bus, and written to the selected I/O port.

Operation: OUT

$I/O((C)) \leftarrow (r)$

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
0	1	r	r	r	0	0	1

The following table shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

TSK80x MCU

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Flag setting (in register F): Flags are not affected.

OUT (n), A

Function: Output to port n

Description: The 8-bit immediate data value n is placed on the low order byte of the address bus. This value is used to select the I/O port (within the lowest 256 bytes of I/O address space) to which an external I/O device is connected. The contents of the Accumulator are placed on the high order byte of the address bus.

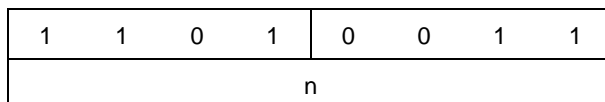
The contents of the Accumulator are read on to the data bus and written to the selected I/O port.

Operation: OUT

$I/O((n)) \leftarrow (A)$

Bytes: 2

Encoding:



Flag setting: (in register F): Flags are not affected.

OUTD

Function: Output and decrement

Description: The byte of data at the memory location specified by the contents of register HL is read and temporarily stored in the CPU. Register B (byte counter) is then decremented and its contents placed on to the high order byte of the address bus. At the same time, the contents of register C are placed onto the low order byte of the address bus. The low order byte of the address (contents of C) is used to select the I/O port (within the lowest 256 bytes of I/O address space) to which an external I/O device is connected.

The temporarily stored byte of data is then placed on the data bus and written, through the selected port, to the peripheral device. The HL register is then decremented.

Operation: OUTD

$I/O((C)) \leftarrow ((HL))$

$(B) \leftarrow (B) - 1$

$(HL) \leftarrow (HL) - 1$

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
1	0	1	0	1	0	1	1

Flag setting (in register F):

S Not Affected
 Z Set if $(B) - 1 = 0$; reset otherwise
 H Not Affected
 P/V Not Affected
 N Set to 1
 C Not affected.

OUTI

Function: Output and increment

Description: The byte of data at the memory location specified by the contents of register HL is read and temporarily stored in the CPU. Register B (byte counter) is then incremented and its contents placed on to the high order byte of the address bus. At the same time, the contents of register C are placed onto the low order byte of the address bus. The low order byte of the address (contents of C) is used to select the I/O port (within the lowest 256 bytes of I/O address space) to which an external I/O device is connected.

The temporarily stored byte of data is then placed on the data bus and written, through the selected port, to the peripheral device. The HL register is then incremented.

Operation: OUTI
 $I/O((C)) \leftarrow ((HL))$
 $(B) \leftarrow (B) - 1$
 $(HL) \leftarrow (HL) + 1$

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
1	0	1	0	0	0	1	1

Flag setting (in register F):

S Not Affected
 Z Set if $(B) - 1 = 0$; reset otherwise
 H Not Affected
 P/V Not Affected
 N Set to 1
 C Not affected.

POP pp

Function: Pop to register

Description: Pops the top two bytes of the external memory stack and loads them into the 16-bit register pp.

The first byte of data from the stack is read from the memory location addressed by the contents of the Stack Pointer register (SP). This data byte is loaded into the low order byte of the register pp. The Stack Pointer is then incremented and the byte of data, at the memory location addressed by the new contents of the SP register, is read and loaded into the high order byte of the pp register. The SP register is then incremented again.

Note that this instruction does not check for stack underflow.

Operation: POP
 $(pp_L) \leftarrow ((SP))$
 $(SP) \leftarrow (SP) + 1$
 $(pp_H) \leftarrow ((SP))$
 $(SP) \leftarrow (SP) + 1$

Bytes: 1

Encoding:

1	1	p	p	0	0	0	1
---	---	---	---	---	---	---	---

The table below shows the registers that pp can represent. The listed 2-bit value for each register replaces the pp entry in the encoding for the instruction.

Register	pp
BC	00
DE	01
HL	10
AF	11

Flag setting (in register F): The flags do not change, except in the case where qq = AF

POP IX

Function: Pop to register

Description: Pops the top two bytes of the external memory stack and loads them into the 16-bit Index register; IX.

The first byte of data from the stack is read from the memory location addressed by the contents of the Stack Pointer register (SP). This data byte is loaded into the low order byte of the IX register. The Stack Pointer is then incremented and the byte of data, at the memory location addressed by the new contents of the SP register, is read and loaded into the high order byte of the IX register. The SP register is then incremented again.

Note that this instruction does not check for stack underflow

Operation POP

$(IX_L) \leftarrow ((SP))$

$(SP) \leftarrow (SP) + 1$

$(IX_H) \leftarrow ((SP))$

$(SP) \leftarrow (SP) + 1$

Bytes 2

Encoding:

1	1	0	1	1	1	0	1
1	1	1	0	0	0	0	1

Flag setting (in register F): Flags are not affected

POP IY

Function: Pop to register

Description: Pops the top two bytes of the external memory stack and loads them into the 16-bit Index register; IY.

The first byte of data from the stack is read from the memory location addressed by the contents of the Stack Pointer register (SP). This data byte is loaded into the low order byte of the IY register. The Stack Pointer is then incremented and the byte of data, at the memory location addressed by the new contents of the SP register, is read and loaded into the high order byte of the IY register. The SP register is then incremented again.

Note that this instruction does not check for stack underflow

Operation POP

$(IY_L) \leftarrow ((SP))$
 $(SP) \leftarrow (SP) + 1$
 $(IY_H) \leftarrow ((SP))$
 $(SP) \leftarrow (SP) + 1$

Bytes 2

Encoding:

1	1	1	1	1	1	0	1
1	1	1	0	0	0	0	1

Flag setting (in register F): Flags are not affected.

PUSH pp

Function: Push to stack

Description: Pushes the contents of the 16-bit register pp to the external memory stack.

The Stack Pointer register (SP) – which contains the current location of the top of the stack – is first decremented. The high order byte of the pp register is then loaded into the memory location addressed by the new contents of the SP register. The Stack Pointer is then decremented again and the low order byte of the pp register loaded into the resulting memory location that is addressed by its contents.

Note that this instruction does not check for stack overflow.

Operation: PUSH

$(SP) \leftarrow (SP) - 1$
 $((SP)) \leftarrow (pp_H)$
 $(SP) \leftarrow (SP) - 1$

$((SP)) \leftarrow (pp_L)$

Bytes: 1

Encoding:

1	1	q	q	0	1	0	1
---	---	---	---	---	---	---	---

The table below shows the registers that pp can represent. The listed 2-bit value for each register replaces the pp entry in the encoding for the instruction.

Registers	pp
BC	00
DE	01
HL	10
AF	11

Flag setting (in register F): The flags are not affected.

PUSH IX

Function: Push to stack

Description: Pushes the contents of the 16-bit Index register IX to the external memory stack.

The Stack Pointer register (SP) – which contains the current location of the top of the stack – is first decremented. The high order byte of the IX register is then loaded into the memory location addressed by the new contents of the SP register. The Stack Pointer is then decremented again and the low order byte of the IX register loaded into the resulting memory location that is addressed by its contents.

Note that this instruction does not check for stack overflow

Operation PUSH

$(SP) \leftarrow (SP) - 1$

$((SP)) \leftarrow (IX_H)$

$(SP) \leftarrow (SP) - 1$

$((SP)) \leftarrow (IX_L)$

Bytes 2

Encoding:

1	1	0	1	1	1	0	1
1	1	1	0	0	1	0	1

Flag setting (in register F): Flags are not affected

PUSH IY

Function: Push to stack

Description: Pushes the contents of the 16-bit Index register IY to the external memory stack. The Stack Pointer register (SP) – which contains the current location of the top of the stack – is first decremented. The high order byte of the IY register is then loaded into the memory location addressed by the new contents of the SP register. The Stack Pointer is then decremented again and the low order byte of the IY register loaded into the resulting memory location that is addressed by its contents.

Note that this instruction does not check for stack overflow

Operation PUSH

$(SP) \leftarrow (SP) - 1$

$((SP)) \leftarrow (IY_H)$

$(SP) \leftarrow (SP) - 1$

$((SP)) \leftarrow (IY_L)$

Bytes 2

Encoding:

1	1	1	1	1	1	0	1
1	1	1	0	0	1	0	1

Flag setting (in register F): Flags are not affected.

RES b, m

Function: Reset bit

Description: Resets bit b of the data from byte variable m. The byte variable m can be any of the following: a register r, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement d.

The bit position within the byte of data (bit0 – bit7) is specified by using 3-bit encoding. The table below shows this encoding for each possible bit position. This 3-bit code replaces the bbb entry in the encoding for the instruction.

Bit position	bbb
0	000
1	001
2	010

Bit position	bbb
3	011
4	100
5	101
6	110
7	111

RES b, r

Operation: RES
 (bit b of (r)) ← 0

Bytes: 2

Encoding:

1	1	0	0	1	0	1	1
1	0	b	b	b	r	r	r

The table below shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

TSK80x MCU

RES b, (HL)

Operation: RES

(bit b of ((HL))) \leftarrow 0

Bytes: 2

Encoding:

1	1	0	0	1	0	1	1
1	0	b	b	b	1	1	0

RES b, (IX+d)

Operation: RES

(bit b of ((IX+d))) \leftarrow 0

Bytes: 4

Encoding:

1	1	0	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
1	0	b	b	b	1	1	0

RES b, (IY+d)

Operation: RES

(bit b of ((IY+d))) \leftarrow 0

Bytes: 4

Encoding:

1	1	1	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
1	0	b	b	b	1	1	0

Flag setting (in register F): Flags are not affected.

RET

Function: Return from subroutine

Description: Pops the top two bytes of the external memory stack and loads them into the Program Counter (PC).

The first byte of data from the stack is read from the memory location addressed by the contents of the Stack Pointer register (SP). This data byte is loaded into the low order byte of the Program Counter. The Stack Pointer is then incremented and the byte of data, at the memory location addressed by the new contents of the SP register, is read and loaded into the high order byte of the PC. The SP register is then incremented again and the processor fetches the new instruction from the address specified by the PC.

Operation: RET

$$(PC_L) \leftarrow ((SP))$$

$$(SP) \leftarrow (SP) + 1$$

$$(PC_H) \leftarrow ((SP))$$

$$(SP) \leftarrow (SP) + 1$$

Bytes: 1

Encoding:

1	1	0	0	1	0	0	1
---	---	---	---	---	---	---	---

Flag setting (in register F): Flags are not affected.

RET cc

Function: Return from subroutine if condition true

Description: Tests condition cc. If it is TRUE, the return address is popped from the external memory stack and loaded into the Program Counter (PC).

The first byte of data from the stack is read from the memory location addressed by the contents of the Stack Pointer register (SP). This data byte is loaded into the low order byte of the Program Counter. The Stack Pointer is then incremented and the byte of data, at the memory location addressed by the new contents of the SP register, is read and loaded into the high order byte of the PC. The SP register is then incremented again and the processor fetches the new instruction from the address specified by the PC.

In the case when the condition is FALSE, the PC is incremented as normal and the next sequential instruction is fetched.

The table below shows the conditions that cc can represent. Each condition tests the state of a flag in the F register. The listed 3-bit value for each condition replaces the ccc entry in the encoding for the instruction.

Condition cc	Description	Relevant flag	ccc
NZ	Non Zero	Z	000
Z	Zero	Z	001
NC	Non Carry	C	010
C	Carry	C	011
PO	Parity odd	P/V	100
PE	Parity even	P/V	101
P	Sign positive	S	110
M	Sign negative	S	111

Operation:

RET

If cc true

$(PC_L) \leftarrow ((SP))$

$(SP) \leftarrow (SP) + 1$

$(PC_H) \leftarrow ((SP))$

$(SP) \leftarrow (SP) + 1$

Else

nothing

Bytes: 1

Encoding:

1	1	c	c	c	0	0	0
---	---	---	---	---	---	---	---

Flag setting (in register F): Flags are not affected.

RETI

Function: Return from maskable interrupt

Description: This instruction can be used as a normal RET instruction (to restore the contents of the Program Counter). Otherwise, it can be used to signal an I/O device that the interrupt routine has been completed.

The RETI instruction facilitates the nesting of interrupts allowing higher priority devices to temporarily suspend the service of lower priority service routines. This instruction does not switch the interrupt enable flip-flop to the enable state. The EI instruction should therefore be used to allow the recognition of interrupts after the completion of the RETI instruction.

Operation: RETI
 $(PC_L) \leftarrow ((SP))$
 $(SP) \leftarrow (SP) + 1$
 $(PC_H) \leftarrow ((SP))$
 $(SP) \leftarrow (SP) + 1$

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
0	1	0	0	1	1	0	1

Flag setting (in register F): Flags are not affected.

RETN

Function: Return from non-maskable interrupt

Description: This instruction is used to return from a non-maskable service routine - to restore the contents of the PC and reconstruct the state of the IFF1 flip-flop (interrupt enable register for maskable interrupts).

In a non-maskable interrupt acknowledge cycle, the contents of IFF1 are loaded to IFF2 and IFF1 is then reset. The RETN instruction reloads the contents of IFF2 back into IFF1. In this way, the original state of maskable interrupts (enabled or disabled) is restored.

Operation: RETN
 $(PC_L) \leftarrow ((SP))$
 $(SP) \leftarrow (SP) + 1$
 $(PC_H) \leftarrow ((SP))$
 $(SP) \leftarrow (SP) + 1$
 $(IFF1) \leftarrow (IFF2)$

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
0	1	0	0	0	1	0	1

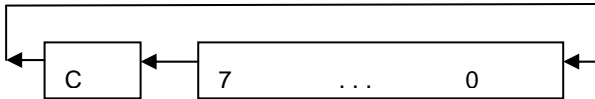
Flag setting (in register F): Flags are not affected.

RL m

Function: Rotate left through Carry

Description: The contents of byte variable m are rotated left by one bit position. The content of bit 7 of m is loaded into the Carry flag (C) and the previous content of the Carry flag is loaded into bit 0 in m. All other bits in m are moved by one position left, as shown in the diagram below.

The byte variable m can be any of the following: a register r, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement d.



RL r

Operation: RL

Bytes: 2

Encoding:

1	1	0	0	1	0	1	1
0	0	0	1	0	r	r	r

The following table shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

RL (HL)

Operation: RL

Bytes: 2

Encoding:

1	1	0	0	1	0	1	1
0	0	0	1	0	1	1	0

RL (IX+d)

Operation: RL

Bytes: 4

Encoding:

1	1	0	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
0	0	0	1	0	1	1	0

RL (IY+d)

Operation: RL

Bytes: 4

TSK80x MCU

Encoding:

1	1	1	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
0	0	0	1	0	1	1	0

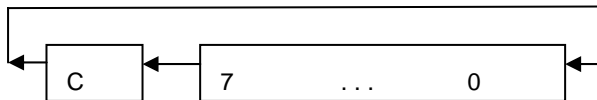
Flag setting (in register F):

S	Set if result is negative; reset otherwise
Z	Set if result is zero; reset otherwise
H	Reset
P/V	Set if parity even; reset otherwise
N	Reset
C	Value of bit 7 of the source data byte.

RLA

Function: Rotate left Accumulator through Carry

Description: The contents of the Accumulator are rotated one bit position left through the Carry flag (C). The content of bit 7 from the Accumulator is loaded into the Carry flag and the previous content of the Carry flag is loaded into bit 0.



Operation: RLA

Bytes: 1

Encoding:

0	0	0	1	0	1	1	1
---	---	---	---	---	---	---	---

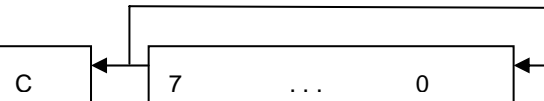
Flag setting (in register F):

S	Not affected
Z	Not affected
H	Reset
P/V	Not affected
N	Reset
C	Value of bit 7 of the Accumulator.

RLC m

Function: Rotate left with Carry

Description: The contents of byte variable m are rotated left by one bit position. The content of bit 7 of m is copied into the Carry flag (C) and also loaded into bit 0 in m. All other bits in m are moved by one position left, as shown in the diagram below.



RLC r

Operation: RLC

Bytes: 2

Encoding:

1	1	0	0	1	0	1	1
0	0	0	0	0	r	r	r

The table below shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

TSK80x MCU

RLC (HL)

Operation: RLC

Bytes: 2

Encoding:

1	1	0	0	1	0	1	1
0	0	0	0	0	1	1	0

RLC (IX+d)

Operation: RLC

Bytes: 4

Encoding:

1	1	0	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
0	0	0	0	0	1	1	0

RLC (IY+d)

Operation: RLC

Bytes: 4

Encoding:

1	1	1	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
0	0	0	0	0	1	1	0

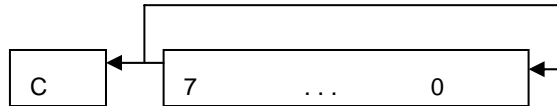
Flag setting (in register F):

- S Set if result is negative; reset otherwise
- Z Set if result is zero; reset otherwise
- H Reset
- P/V Set if parity even; reset otherwise
- N Reset
- C Value of bit 7 of the source data byte (same as new bit 0).

RLCA

Function: Rotate left Accumulator with Carry

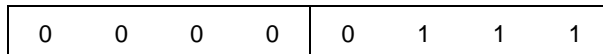
Description: The contents of the Accumulator are rotated one bit position left. The content of bit 7 from the Accumulator is copied into the Carry flag and is also loaded into bit 0.



Operation: RLCA

Bytes: 1

Encoding:



Flag setting (in register F):

S	Not affected
Z	Not affected
H	Reset
P/V	Not affected
N	Reset
C	Value of bit 7 of the Accumulator (same as new bit 0).

RLD

Function: Rotate left digit

Description: The byte of data from memory at the location addressed by the contents of the HL register is read and the rotation proceeds as follows:

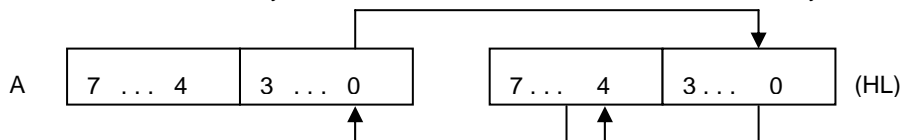
The contents of the four low order bits are moved into the four high order bits.

The previous contents of the four high order bits of this byte are moved into the four low order bits of the Accumulator.

The previous contents of the four low order bits of the Accumulator are moved into the four low order bits of the byte read from memory.

The contents of the four high order bits of the Accumulator remain unchanged.

The modified byte of data is stored at the same location in memory.



Operation: RLD

Bytes: 2

TSK80x MCU

Encoding:

1	1	1	0	1	1	0	1
0	1	1	0	1	1	1	1

Flag setting (in register F):

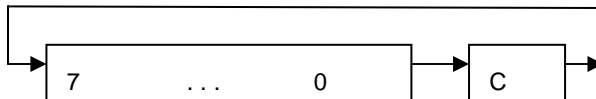
S	Set if result in Accumulator is negative; reset otherwise
Z	Set if result in Accumulator is zero; reset otherwise
H	Reset
P/V	Set if parity of Accumulator is even after operation; reset otherwise
N	Reset
C	Not affected.

RR m

Function: Rotate right through Carry

Description: The contents of byte variable *m* are rotated right by one bit position. The content of bit 0 of *m* is loaded into the Carry flag (C) and the previous content of the Carry flag is loaded into bit 7 in *m*. All other bits in *m* are moved by one position right, as shown in the diagram below.

The byte variable *m* can be any of the following: a register *r*, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement *d*.



RR r

Operation: RR

Bytes: 2

Encoding:

1	1	0	0	1	0	1	1
0	0	0	1	1	r	r	r

The table below shows the registers that *r* can represent. The listed 3-bit value for each register replaces the *rrr* entry in the encoding for the instruction.

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

RR (HL)

Operation: RR

Bytes: 2

Encoding:

1	1	0	0	1	0	1	1
0	0	0	1	1	1	1	0

RR (IX+d)

Operation: RR

Bytes: 4

Encoding:

1	1	0	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
0	0	0	1	1	1	1	0

RR (IY+d)

Operation: RR

Bytes: 4

TSK80x MCU

Encoding:

1	1	1	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
0	0	0	1	1	1	1	0

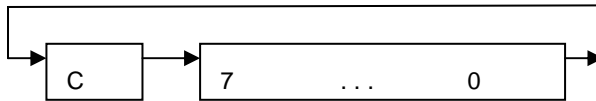
Flag setting (in register F):

S	Set if result is negative; reset otherwise
Z	Set if result is zero; reset otherwise
H	Reset
P/V	Set if parity even; reset otherwise
N	Reset
C	Value of bit 0 of source byte.

RRA

Function: Rotate right Accumulator through Carry

Description: The contents of the Accumulator are rotated one bit position right through the Carry flag (C). The content of bit 0 from the Accumulator is loaded into the Carry flag and the previous content of the Carry flag is loaded into bit 7.



Operation: RRA

Bytes: 1

Encoding:

0	0	0	1	1	1	1	1
---	---	---	---	---	---	---	---

Flag setting (in register F):

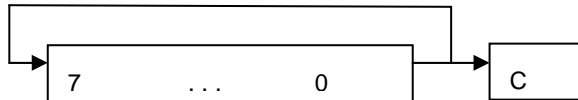
S	Not affected
Z	Not affected
H	Reset
P/V	Not affected
N	Reset
C	Value of bit 0 of the Accumulator.

RRC m

Function: Rotate right with Carry

Description: The contents of byte variable m are rotated right by one bit position. The content of bit 0 of m is copied into the Carry flag (C) and also loaded into bit 7 in m. All other bits in m are moved by one position right, as shown in the diagram below.

The byte variable m can be any of the following: a register r, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement d.



RRC r

Operation: RRC

Bytes: 2

Encoding:

1	1	0	0	1	0	1	1
0	0	0	0	1	r	r	r

The table below shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

RRC (HL)

Operation: RRC

Bytes: 2

Encoding:

TSK80x MCU

1	1	0	0	1	0	1	1
0	0	0	0	1	1	1	0

RRC (IX+d)

Operation: RRC

Bytes: 4

Encoding:

1	1	0	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
0	0	0	0	1	1	1	0

RRC (IY+d)

Operation: RRC

Bytes: 4

Encoding:

1	1	1	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
0	0	0	0	1	1	1	0

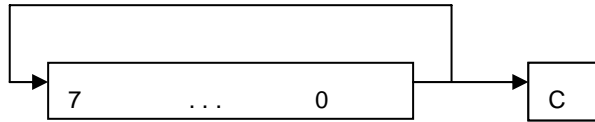
Flag setting (in register F):

- S Set if result is negative; reset otherwise
- Z Set if result is zero; reset otherwise
- H Reset
- P/V Set if parity even; reset otherwise
- N Reset
- C Value of bit 0 of the source data byte (same as new bit 7).

RRCA

Function: Rotate right Accumulator with Carry

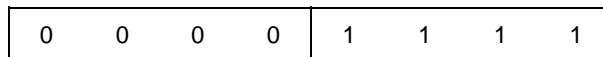
Description: The contents of the Accumulator are rotated one bit position right. The content of bit 0 from the Accumulator is copied into the Carry flag and is also loaded into bit 7.



Operation: RLCA

Bytes: 1

Encoding:



Flag setting (in register F):

S	Not affected
Z	Not affected
H	Reset
P/V	Not affected
N	Reset
C	Value of bit 0 of the Accumulator (same as new bit 7)

RRD

Function: Rotate right digit

Description: The byte of data from memory at the location addressed by the contents of the HL register is read and the rotation proceeds as follows:

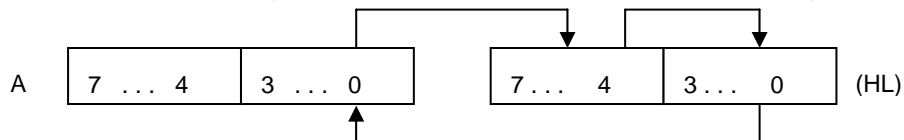
The contents of the four low order bits are moved into the four low order bits of the Accumulator.

The previous contents of the four low order bits of the Accumulator are moved into the four high order bits of the data byte.

The previous contents of the four high order bits of the data byte are moved into the four low order bits of the byte.

The contents of the four high order bits of the Accumulator remain unchanged.

The modified byte of data is stored at the same location in memory.



TSK80x MCU

Operation: RRD

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
0	1	1	0	0	1	1	1

Flag setting (in register F):

S	Set if result in Accumulator is negative; reset otherwise
Z	Set if result in Accumulator is zero; reset otherwise
H	Reset
P/V	Set if parity of Accumulator is even after operation; reset otherwise
N	Reset
C	Not affected.

RST p

Function: Restart to p

Description: Pushes the current contents of the Program Counter (PC) to the external memory stack and then loads the page zero memory location specified by the variable p, into the PC.

The Stack Pointer register (SP) – which contains the current location of the top of the stack – is first decremented. The high order byte of the PC register is then loaded into the memory location addressed by the new contents of the SP register. The Stack Pointer is then decremented again and the low order byte of the PC register loaded into the resulting memory location that is addressed by its contents.

The page zero memory location is then loaded into the PC, with 00h loaded into the high order byte and the value p loaded into the low order byte. The next instruction is fetched from this new address.

The variable p describes one of eight possible addresses to jump to and is specified using 3-bit encoding. The table below shows this encoding for each possible jump address. This 3-bit code replaces the ttt entry in the encoding for the instruction.

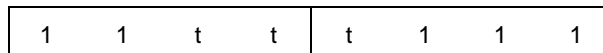
p	ttt
00h	000
08h	001
10h	010
18h	011

p	ttt
20h	100
28h	101
30h	110
38h	111

Operation: RST
 $(SP) \leftarrow (SP) - 1$
 $((SP)) \leftarrow (PC_H)$
 $(SP) \leftarrow (SP) - 1$
 $((SP)) \leftarrow (PC_L)$
 $(PC_H) \leftarrow 00h$
 $(PC_L) \leftarrow p$

Bytes: 1

Encoding:



Flag setting (in register F): Flags are not affected.

SBC A, s

Function: Subtract 8-bit with carry

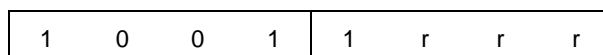
Description: Subtracts the data from the byte variable s and the Carry flag from the contents of the Accumulator and stores the result in the Accumulator. The byte variable s can be any of the following: a register r, an immediate data value n, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement d.

SBC A, r

Operation: SBC
 $(A) \leftarrow (A) - (r) - (C)$

Bytes: 1

Encoding:



TSK80x MCU

The table below shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

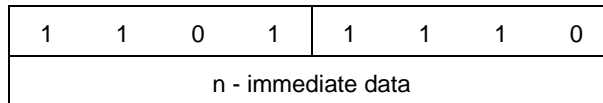
SBC A, n

Operation: SBC

$$(A) \leftarrow (A) - n - (C)$$

Bytes: 2

Encoding:



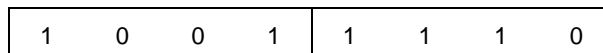
SBC A, (HL)

Operation: SBC

$$(A) \leftarrow (A) - ((HL)) - (C)$$

Bytes: 1

Encoding:



SBC A, (IX+d)

Operation: SBC

$$(A) \leftarrow (A) - ((IX+d)) - (C)$$

Bytes: 3

Encoding:

1	1	0	1	1	1	0	1
1	0	0	1	1	1	1	0
d							

SBC A, (IY+d)

Operation: SBC

$$(A) \leftarrow (A) - ((IY+d)) - (C)$$

Bytes: 3

Encoding:

1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	0
d							

Flag setting (in register F):

- S Set if result is negative; reset otherwise
- Z Set if result is zero; reset otherwise
- H Set if borrow from bit 4; reset otherwise
- P/V Set if overflow; reset otherwise
- N Set
- C Set if borrow; reset otherwise.

SBC HL, pp

Function: Subtract 16-bit with carry

Description: Subtracts the contents of register pp and the Carry flag from the contents of the HL register. The result is stored in the HL register.

Operation: SBC

$$(HL) \leftarrow (HL) - (pp) - (C)$$

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
0	1	p	p	0	0	1	0

The table below shows the registers that pp can represent. The listed 2-bit value for each register replaces the pp entry in the encoding for the instruction.

Register	pp
BC	00
DE	01
HL	10
SP	11

Flag setting (in register F):

- S Set if result is negative; reset otherwise
- Z Set if result is zero; reset otherwise
- H Set if borrow from bit 12; reset otherwise
- P/V Set if overflow; reset otherwise
- N Set
- C Set if borrow; reset otherwise.

SCF

Function: Set Carry flag

Description: Sets the Carry flag in register F (flag register).

Operation: SCF

(C) ← 1

Bytes: 1

Encoding:

0	0	1	1	0	1	1	1
---	---	---	---	---	---	---	---

Flag setting (in register F):

- S Not affected
- Z Not affected
- H Reset
- P/V Not affected
- N Reset
- C Set

SET b, m

Function: Set bit

Description: Sets bit b of the data from byte variable m. The byte variable m can be any of the following: a register r, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement d.

The bit position within the byte of data (bit0 – bit7) is specified by using 3-bit encoding. The table below shows this encoding for each possible bit position. This 3-bit code replaces the bbb entry in the encoding for the instruction.

Bit position	bbb
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

SET b, r

Operation: SET
(bit b of (r)) ← 1

Bytes: 2

Encoding:

1	1	0	0	1	0	1	1
1	1	b	b	b	r	r	r

The following table shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

TSK80x MCU

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

SET b, (HL)

Operation: SET
(bit b of ((HL))) ← 1

Bytes: 2

Encoding:

1	1	0	0	1	0	1	1
1	1	b	b	b	1	1	0

SET b, (IX+d)

Operation: SET
(bit b of ((IX+d))) ← 1

Bytes: 4

Encoding:

1	1	0	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
1	1	b	b	b	1	1	0

SET b, (IY+d)

Operation: SET
(bit b of ((IY+d))) ← 1

Bytes: 4

Encoding:

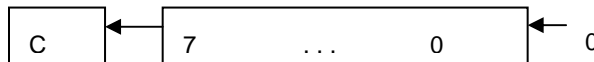
1	1	1	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
1	1	b	b	b	1	1	0

Flag setting (in register F): Flags are not affected.

SLA m

Function: Shift left arithmetically

Description: The contents of byte variable m are arithmetically shifted left by one bit position. The content of bit 7 of m is copied into the Carry flag (C). All other bits in m are moved by one position left, with a zero being loaded into bit 0, as shown in the diagram below. The byte variable m can be any of the following: a register r, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement d.



SLA r

Operation: SLA

Bytes: 2

Encoding:

1	1	0	0	1	0	1	1
0	0	1	0	0	r	r	r

The table below shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

TSK80x MCU

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

SLA (HL)

Operation: SLA

Bytes: 2

Encoding:

1	1	0	0	1	0	1	1
0	0	1	0	0	1	1	0

SLA (IX+d)

Operation: SLA

Bytes: 4

Encoding:

1	1	0	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
0	0	1	0	0	1	1	0

SLA (IY+d)

Operation: SLA

Bytes: 4

Encoding:

1	1	1	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
0	0	1	0	0	1	1	0

Flag setting (in register F):

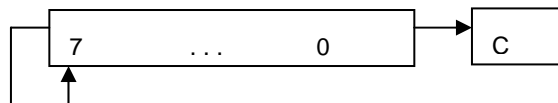
- S Set if result is negative; reset otherwise
- Z Set if result is zero; reset otherwise
- H Reset
- P/V Set if parity even; reset otherwise
- N Reset
- C Value of bit 7.

SRA m

Function: Shift right arithmetically

Description: The contents of byte variable m are arithmetically shifted right by one bit position. The content of bit 7 of m is copied back into itself (and so remains unchanged), while bit 0 is copied into the Carry flag (C). All other bits in m are moved by one position right, as shown in the diagram below.

The byte variable m can be any of the following: a register r, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement d.



SRA r

Operation: SRA

Bytes: 2

Encoding:

1	1	0	0	1	0	1	1
0	0	1	0	1	r	r	r

TSK80x MCU

The table below shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

SRA (HL)

Operation: SRA

Bytes: 2

Encoding:

1	1	0	0	1	0	1	1
0	0	1	0	1	1	1	0

SRA (IX+d)

Operation: SRA

Bytes: 4

Encoding:

1	1	0	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
0	0	1	0	1	1	1	0

SRA (IY+d)

Operation: SRA

Bytes: 4

Encoding:

1	1	1	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
0	0	1	0	1	1	1	0

Flag setting (in register F):

- S Set if result is negative; reset otherwise
- Z Set if result is zero; reset otherwise
- H Reset
- P/V Set if parity even; reset otherwise
- N Reset
- C Value of bit 0 of the source data byte

SRL m

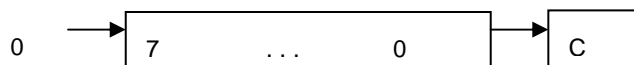
Function:

Shift right logically

Description:

The contents of byte variable m are logically shifted right by one bit position. A zero is loaded into bit 7. All other bits in m are moved by one position right, with bit 0 being copied into the Carry flag (C), as shown in the diagram below.

The byte variable m can be any of the following: a register r, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement d.



SRL r

Operation:

SRL

Bytes:

2

Encoding:

1	1	0	0	1	0	1	1
0	0	1	1	1	r	r	r

The table below shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

TSK80x MCU

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

SRL (HL)

Operation: SRL

Bytes: 2

Encoding:

1	1	0	0	1	0	1	1
0	0	1	1	1	1	1	0

SRL (IX+d)

Operation: SRL

Bytes: 4

Encoding:

1	1	0	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
0	0	1	1	1	1	1	0

SRL (IY+d)

Operation: SRL

Bytes: 4

Encoding:

1	1	1	1	1	1	0	1
1	1	0	0	1	0	1	1
d							
0	0	1	1	1	1	1	0

Flag setting (in register F):

- S Reset
- Z Set if result is zero; reset otherwise
- H Reset
- P/V Set if parity even; reset otherwise
- N Reset
- C Value of bit 0 of source data byte

SUB s

Function: Subtract 8-bit data

Description: Subtracts the data from the byte variable s from the Accumulator and stores the result in the Accumulator. The byte variable s can be any of the following: a register r, an immediate data value n, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement d.

SUB r

Operation: SUB

$$(A) \leftarrow (A) - (r)$$

Bytes: 1

Encoding:

1	0	0	1	0	r	r	r
---	---	---	---	---	---	---	---

The following table shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

TSK80x MCU

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

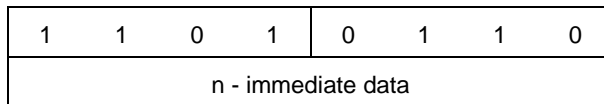
SUB n

Operation: SUB

$$(A) \leftarrow (A) - n$$

Bytes: 2

Encoding:



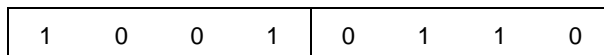
SUB (HL)

Operation: SUB

$$(A) \leftarrow (A) - ((HL))$$

Bytes: 1

Encoding:



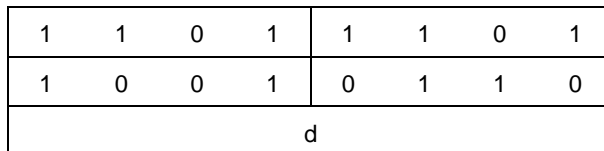
SUB (IX+d)

Operation: SUB

$$(A) \leftarrow (A) - ((IX+d))$$

Bytes: 3

Encoding:



SUB (IY+d)

Operation: SUB
 $(A) \leftarrow (A) - ((IY+d))$

Bytes: 3

Encoding:

1	1	1	1	1	1	0	1
1	0	0	1	0	1	1	0
d							

Flag setting (in register F):

S Set if result is negative; reset otherwise
 Z Set if result is zero; reset otherwise
 H Set if borrow from bit 4; reset otherwise
 P/V Set if overflow; reset otherwise
 N Set
 C Set if borrow; reset otherwise

TRAP (TSK80A_D only)

Function: Trap instruction

Description: The function of this instruction depends on the state of the debugenable signal. If debugenable is not active, the instruction operates as a NOP instruction (do nothing). When debugenable is active, the TRAP instruction forces the processor to stop and go to debug mode. In this case, the Program Counter is prohibited from incrementing, resulting in the TRAP instruction being fetched all the time, until the debugenable signal becomes inactive again.

Operation: TRAP

Bytes: 2

Encoding:

1	1	1	0	1	1	0	1
0	0	0	0	0	0	0	0

Flag setting (in register F): Flags are not affected

XOR s

Function: Logical XOR

Description: Performs a logical XOR between the data from byte variable s and the contents of the Accumulator. The byte variable s can be any of the following: a register r, an immediate data value n, a data value in memory at a location selected by the contents of the HL register, or data from memory at the location selected by the sum of an Index register (IX or IY) and an 8-bit displacement d.

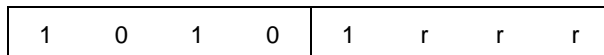
XOR r

Operation: XOR

$$(A) \leftarrow (A) \vee (r)$$

Bytes: 1

Encoding:



The table below shows the registers that r can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for the instruction.

Register	rrr
B	000
C	001
D	010
E	011
H	100
L	101
A	111

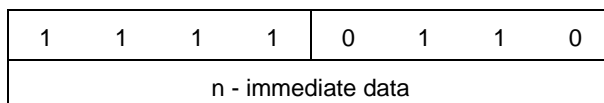
XOR n

Operation: XOR

$$(A) \leftarrow (A) \vee n$$

Bytes: 2

Encoding:



XOR (HL)

Operation: XOR
 $(A) \leftarrow (A) \vee ((HL))$

Bytes: 1

Encoding:

1	0	1	0	1	1	1	0
---	---	---	---	---	---	---	---

XOR (IX+d)

Operation: XOR
 $(A) \leftarrow (A) \vee ((IX+d))$

Bytes: 3

Encoding:

1	1	0	1	1	1	0	1
1	0	1	0	1	1	1	0
d							

XOR (IY+d)

Operation: XOR
 $(A) \leftarrow (A) \vee ((IY+d))$

Bytes: 3

Encoding:

1	1	1	1	1	1	0	1
1	0	1	0	1	1	1	0
d							

Flag setting (in register F):

- S Set if result is negative; reset otherwise
- Z Set if result is zero; reset otherwise
- H Reset
- P/V Set if parity even; reset otherwise
- N Reset
- C Reset.

Instruction timing

The timing diagram of Figure 17 illustrates the signal timing involved in the execution (in sequence) of the following three instructions – NOP, INC (HL) and OUT (C), B.

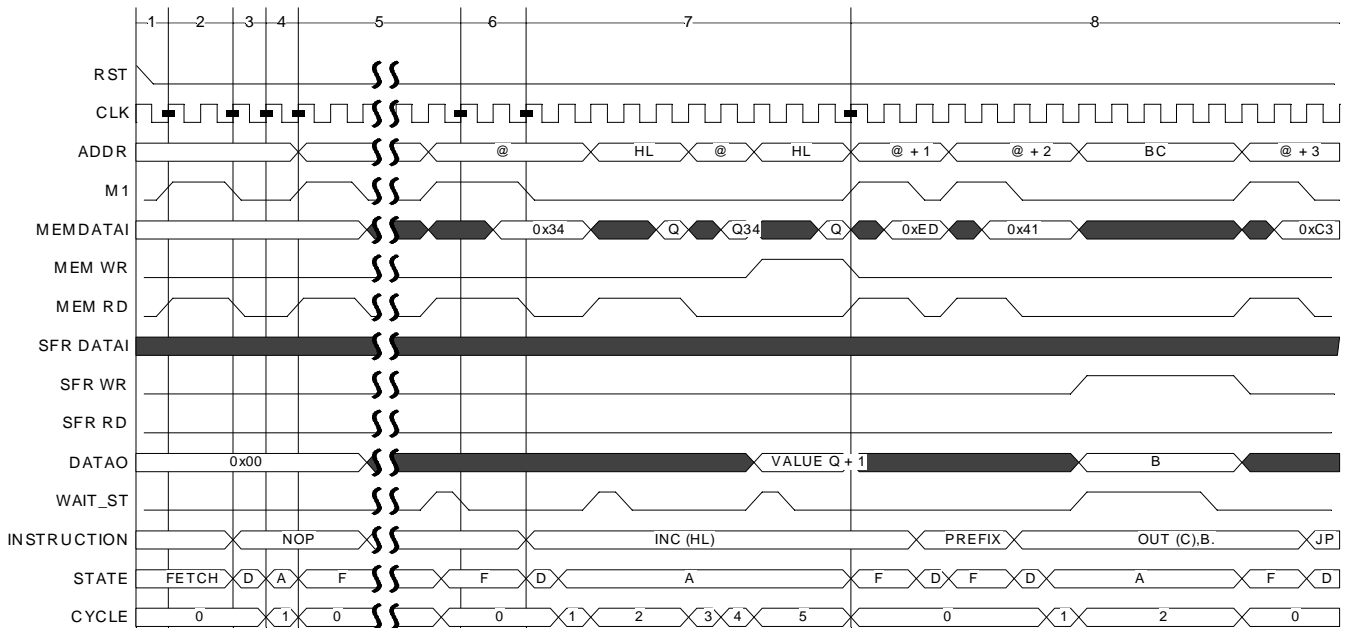


Figure 17. Example instruction cycle, including read and write

The timing diagram covers eight key regions of activity, summarized by the corresponding numbered entries that follow:

- 1 - An external synchronous reset is received on the RST input. When the reset line becomes inactive, the processor starts executing
- 2 - Output signals M1 and MEMRD go active. Signal M1 going High characterizes a FETCH, and the instruction at address 0000h is retrieved
- 3 - The processor moves into the DECODE state. The Opcode loaded into the instruction register is used to determine the number of cycles required by the instruction
- 4 - The processor enters the ACTION state. The number of cycles required to complete execution of the instruction is determined by the Decoder and passed to the state machine. Cycles are incremented using the state machine's internal cycle counter until this defined value is reached. The Decoder generates all internal signals required for each cycle of the current instruction
- 5 - The NOP instruction has been executed – the processor proceeds to fetch a new instruction and continue its execution of program code
- 6 - Fetching of the new instruction is delayed due to WAIT_ST signal being active
- 7 - The input data is valid and the Opcode is loaded into the instruction register. The instruction (INC (HL)) loads the value from location (HL) in memory, increments the value and then stores

the new value back at the same address. To perform this instruction, five ACTION cycles are required.

- In cycle 2, the processor reads the value at (HL) – note that MEMRD is active for the read, while M1 is inactive because it is not a FETCH
 - In cycles 3 and 4, the value is incremented
 - In cycle 5, the processor writes the new value back to memory, at (HL). MEMWR is active for the write
- 8 - INC (HL) is complete and the processor loads the next new instruction – OUT (C), B. This instruction stores the current value in internal register B, into the IO space location (C). The instruction takes two ACTION cycles. In cycle 2, the value of the register is output on the DATAO bus, and the SFRWR signal is taken active – indicating a write transaction to the external IO space. The address bus (ADDR) value is (BC), even if the IO space is only 256 bytes wide

Revision History

Date	Version No.	Revision
22-Jan-2004	1.0	New product release
09-Dec-2004	1.1	Updated to reflect new TSK80 Processor core
08-Feb-2005	1.2	Modifications to debug panel information in On-Chip Debugging section.
09-May-2005	1.3	Updated for SP4
22-Aug-2005	1.4	Minor Image Enhancements
12-Dec-2005	1.5	Path references updated for Altium Designer 6
28-Aug-2006	1.6	Fixed heading levels in Instruction Set section.

Software, hardware, documentation and related materials:

Copyright © 2006 Altium Limited.

All rights reserved. You are permitted to print this document provided that (1) the use of such is for personal use only and will not be copied or posted on any network computer or broadcast in any media, and (2) no modifications of the document is made. Unauthorized duplication, in whole or part, of this document by any means, mechanical or electronic, including translation into another language, except for brief excerpts in published reviews, is prohibited without the express written permission of Altium Limited. Unauthorized duplication of this work may also be prohibited by local statute. Violators may be subject to both criminal and civil penalties, including fines and/or imprisonment. Altium, Altium Designer, Board Insight, CAMtastic, CircuitStudio, Design Explorer, DXP, LiveDesign, NanoBoard, NanoTalk, Nexar, nVisage, P-CAD, Protel, SimCode, Situs, TASKING, and Topological Autorouting and their respective logos are trademarks or registered trademarks of Altium Limited or its subsidiaries. All other registered or unregistered trademarks referenced herein are the property of their respective owners and no trademark rights to the same are claimed.