TSK51x MCU

Summary

Core Reference
CR0115 (v2.0) March 13, 2008

The TSK51x is a fully functional, 8-bit microcontroller, incorporating the Harvard architecture. This core reference includes architectural and hardware descriptions, instruction sets and on-chip debugging functionality for the TSK51x family.

The TSK51x is the core of a fast, single-chip, 8-bit microcontroller, which executes all ASM51 instructions and is instruction set compatible with the 80C31. The TSK51x serves software and hardware interrupts, provides an interface for serial communications and incorporates a timer system.

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 operations
  - 8 bit logical operations
  - Boolean manipulations
  - 8 x 8 bit multiplication
  - 8 / 8 bit division.
- 32-bit Input/Output ports
  - Four 8-bit I/O ports
- Two 16-bit Timer/Counters
- Serial Peripheral Interfaces in full duplex mode
  - Synchronous mode, fixed baud rate
  - 8-bit UART mode, variable baud rate
  - 9-bit UART mode, fixed baud rate
  - 9-bit UART mode, variable baud rate
  - Multiprocessor communication.
- Interrupt Controller
  - Two Priority Levels
  - Five interrupt sources.
- Internal memory interface
  - Can address up to 64KB of Internal Program memory space.
  - Can address up to 256 bytes of Read/Write Data memory Space.
- External memory interface
  - Can address up to 64KB of External Program memory Space
  - Can address up to 64KB of External Data memory Space.
- Special Function Registers interface
  - Services up to 107 External Special Function Registers

Available Devices

Both standard and debug-enabled (OCD) versions of the microcontroller are available – the TSK51A and TSK51A_D respectively. These devices can be found in the FPGA Processors integrated library (`FPGA Processors.IntLib`), located in the `\Library\Fpga` folder of the installation.

Architectural Overview

Symbols

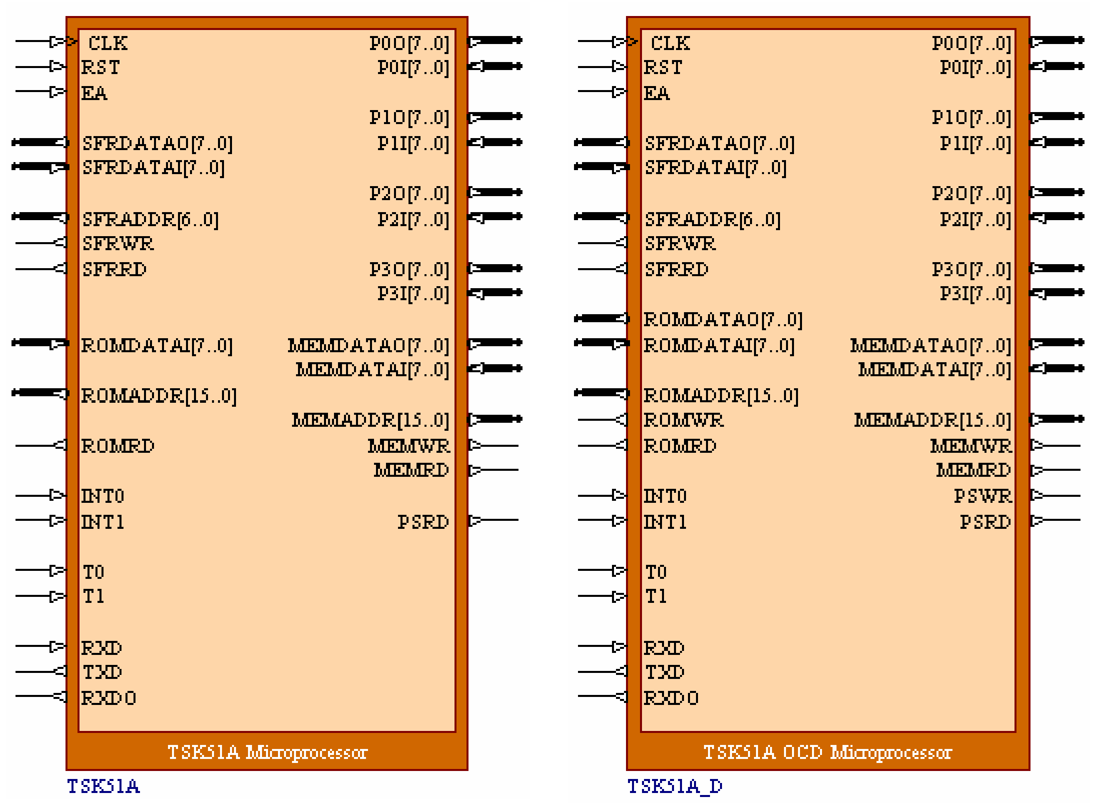


Figure 1. TSK51x family symbols

Pin Description

The pinout of the TSK51x has not been fixed to any specific device I/O - allowing flexibility with user application. The TSK51x contains only unidirectional pins - inputs or outputs.

Table 1. TSK51x Pin description

Name	Type	Polarity/Bus size	Description
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 two clock cycles while the external system clock (CLK) is running resets the device.
EA	I	High	External Access Enable. EA must be externally held High to enable the device to fetch code from external Program memory (0000h - FFFFh). If EA is held Low, the device executes from internal Program memory unless the Program Counter contains an address greater than 0FFFh.
External Special Function Registers Interface Signals			
SFRDATAO	O	8	SFR data bus output

Name	Type	Polarity/Bus size	Description
SFRDATAI	I	8	SFR data bus input
SFRADDR	O	7	SFR address bus
SFRWR	O	High	SFR write enable
SFRRD	O	High	SFR output enable
Internal Program Memory Interface Signals			
ROMDATAI	I	8	Memory data bus input
ROMDATAO[1]	O	8	Memory data bus output
ROMADDR	O	16	Memory address bus
ROMWR[1]	O	High	Memory write enable
ROMRD	O	High	Memory output enable
Interrupt Signals			
INT0	I	Rise/High	External interrupt 0. Interrupt type (rising edge or High level) is determined by setting or clearing bit 0 (IT0) in the TCON register, respectively
INT1	I	Rise/High	External interrupt 1. Interrupt type (rising edge or High level) is determined by setting or clearing bit 2 (IT1) in the TCON register, respectively
Timer Signals			
T0	I	Fall	Timer 0 external clock input
T1	I	Fall	Timer 1 external clock input
Serial Interface Signals			
RXD	I	-	Serial port 0 input (receive)
TXD	O	-	Serial port 0 output (transmit)
RXDO	O	-	Serial port 0 output (transmit in Mode 0)
I/O Port Interface Signals			
P0O P0I	O I	8 8	**Port 0** is an 8-bit bi-directional I/O port with separated inputs and outputs.
P1O P1I	O I	8 8	**Port 1** is an 8-bit bi-directional I/O port with separated inputs and outputs.
P2O P2I	O I	8 8	**Port 2** is an 8-bit bi-directional I/O port with separated inputs and outputs.
P3O P3I	O I	8 8	**Port 3** is an 8-bit bi-directional I/O port with separated inputs and outputs.
External Memory Interface Signals			
MEMDATAO	O	8	External memory output
MEMDATAI	I	8	External memory input

[1] TSK51A_D only

Name	Type	Polarity/Bus size	Description
MEMADDR	O	16	External address bus
MEMWR	O	High	External Data memory write enable
MEMRD	O	High	External Data memory output enable
PSWR[2]	O	High	External Program memory write enable
PSRD	O	High	External Program memory output enable

Memory Organization

Memory in the TSK51x is organized into three distinct areas:

- Program memory (internal ROM or external ROM)
- External Data memory (external RAM)
- Internal Data memory (internal RAM).

Program Memory

The TSK51x can address up to 64KB of Program memory, implemented as either internal ROM, external ROM, or a combination of both.

After a reset has been issued, the CPU starts program execution from location 0000h.

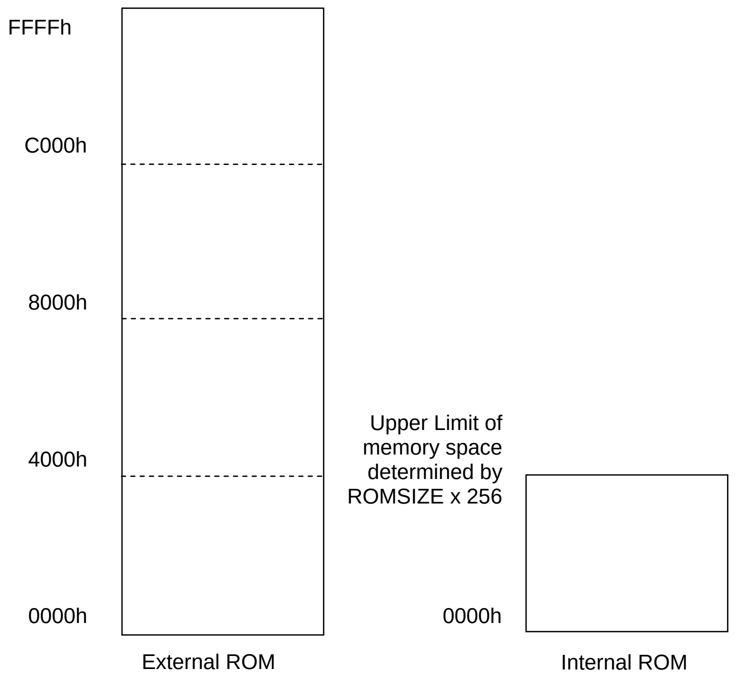


Figure 2. Program memory map

Up to 64KB of internal Program memory space can be addressed. The actual size of the memory space is determined by the value stored in the ROMSIZE register (see 0) and is calculated as:

Internal Program memory = ROMSIZE x 256

The size of internal Program memory is therefore under direct control of software. By default, after a reset, the ROMSIZE register contains the value 10h, which yields a memory space of 4KB. To increase or decrease this size, simply load the ROMSIZE register with the appropriate value.

[2] TSK51A_D only

Program code can be fetched from external or internal Program memory. This selection is made by strapping pin EA (External Address) to VCC or GND respectively. Note that EA can be changed at anytime whilst the processor is running, giving full control over the particular memory space used.

If EA is held High, all the program code is fetched from external memory. If EA is held Low, the lowest n bytes of program code is fetched from internal ROM, where n is the result of ROMSIZE x 256.

When the extent of internal memory space is reached, program code will then automatically be fetched from external memory space. The Program Counter is not reset however, so code will be fetched from the next memory address, but within external memory space.

If the ROMSIZE register contains 00h, the fetch will automatically default to the external Program memory, even if EA is Low.

The lower part of the Program memory includes interrupt and reset vectors. The interrupt vectors are spaced at 8-byte intervals, starting from 0003h for External Interrupt 0.

Table 2. Reset vectors

Location	Service
0003h	External Interrupt 0
000Bh	Timer 0 overflow
0013h	External Interrupt 1
001Bh	Timer 1 overflow
0023h	Serial Port Interrupt

These locations may be used for program code, if the corresponding interrupts are not used (disabled).

When using Internal Program memory, a separate block is placed in the design – external to the component symbol for the core. With the standard version of the core (TSK51A), a block of ROM is used, the size of which depends on the requirements of the design. With the OCD version (TSK51A_D), because this version of the core allows you to write to Program memory space, RAM must be used instead, as shown in Figure 3.

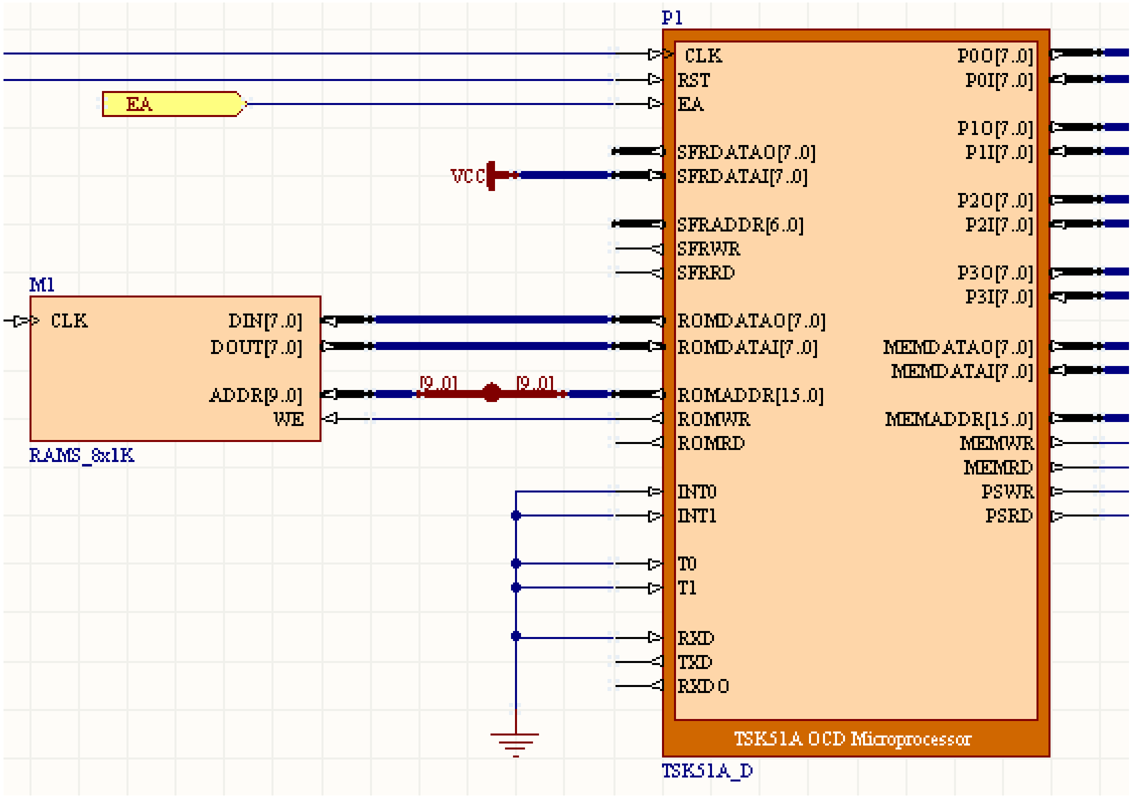


Figure 3. Using RAM for TSK51A_D internal Program memory

RAM and ROM blocks can be found in the FPGA Memories integrated library (`\Library\Fpga\FPGA Memories.IntLib`).

Data Memory

External Data Memory

The TSK51x Microcontroller core incorporates the Harvard architecture, with separate program (code) and data spaces:

- The code from external Program memory is fetched by strobing the PSRD pin.
- Data is read from external Data memory by strobing the MEMRD pin and written to external Data memory by strobing the MEMWR pin.
- The external Data memory space can be accessed directly, through the 16 bit Data Pointer Register (DPTR), or indirectly, using register R0 or R1 and the external Data memory paging register, XP.
- Data is read back on the MEMDATAI bus.

Internal Data Memory

The TSK51x has a 256 byte block of RAM dedicated for use as internal Data memory. This RAM cannot be upgraded in size. The internal Data memory interface is therefore not exposed to the user through the schematic symbol.

The 256 bytes of memory space (00h to FFh) can be accessed by either direct or indirect addressing (where supported). An internal Data memory address is always 1 byte in width.

The upper 128 bytes contain the Special Function Registers (SFRs). This area of internal Data memory is accessible only by direct addressing.

The lower 128 bytes contain work registers and bit-addressable memory. The lower 48 bytes of this area of memory space are further divided as follows:

- The lower 32 bytes (00h – 1Fh) form four banks of eight registers (R0-R7). The RS0 and RS1 bits in the Program Status Word register (PSW) select which bank is currently in use.
- The next 16 bytes (20h – 2Fh) form a block of bit-addressable memory space, covering the bit address range 00h-7Fh.

All of the bytes in this lower half of the internal Data memory space are accessible through direct or indirect addressing.

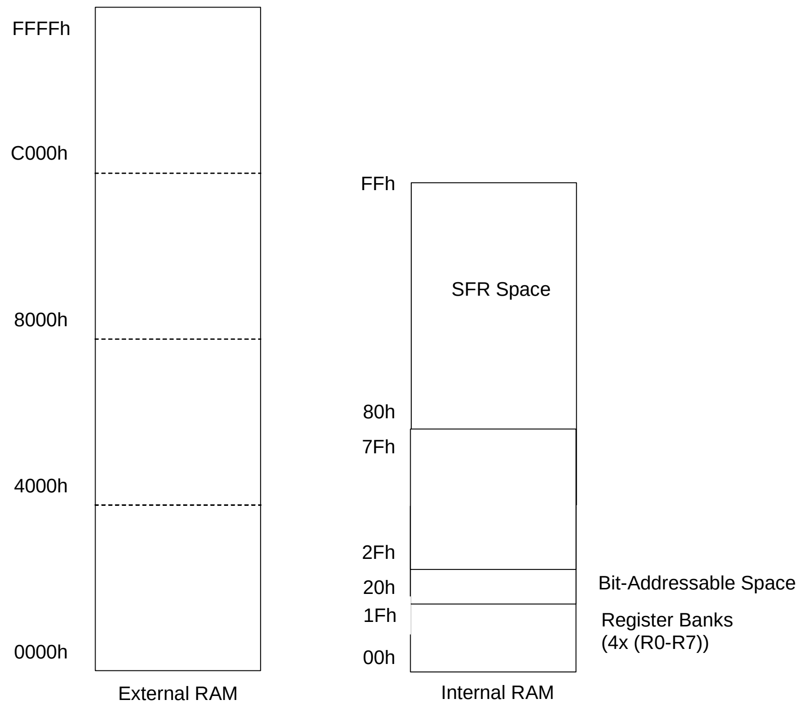


Figure 4. Data memory map

Special Function Registers

A map of the Special Function Registers is shown in Table 3. Only a few addresses are occupied, the others are not implemented. Read access to unimplemented addresses will return undefined data, while writing to them will have no effect.

Table 3. Special Function Registers location

Hex\Bin	**X000**	**X001**	**X010**	**X011**	**X100**	**X101**	**X110**	**X111**	**Bin/Hex**
F8									**FF**
F0	B								**F7**
E8									**EF**
E0	ACC								**E7**
D8									**DF**
D0	PSW								**D7**
C8									**CF**
C0									**C7**
B8	IP								**BF**
B0	P3								**B7**
A8	IE								**AF**
A0	P2								**A7**
98	SCON	SBUF						XP	**9F**
90	P1								**97**
88	TCON	TMOD	TL0	TL1	TH0	TH1		ROMSIZE	**8F**
80	P0	SP	DPL	DPH				PCON	**87**

Accumulator (ACC)

Most instructions use the Accumulator to hold the operand. Note that the mnemonics for Accumulator-specific instructions refer to the Accumulator as A, not ACC.

B register

The B register is used during multiply and divide instructions. It can also be used as a scratch-pad register to hold temporary data.

External Data memory Paging Register (XP)

The content of the XP register is loaded onto the high order byte of the external memory address bus (MEMADDR) during a MOVX @Ri instruction. The XP register is used to implement paging and can provide access to up to 256 pages in external Data memory. Each page can contain up to 256 bytes of data – dependent on the contents of the register Ri. Therefore the maximum addressable Data memory space is 64KB.

Program Status Word Register (PSW)

Table 4. PSW register flags

MSB							LSB
CY	AC	F1	RS1	RS0	OV	F0	P

Table 5. PSW register bit functions

Bit	Symbol	Function
PSW.7	CY	Carry flag
PSW.6	AC	Auxiliary Carry flag for BCD operations
PSW.5	F1	General purpose Flag 1 available for user
PSW.4	RS1	Register bank select control bit 1, used to select working register bank
PSW.3	RS0	Register bank select control bit 0, used to select working register bank
PSW.2	OV	Overflow flag
PSW.1	F0	General purpose Flag 0 available for user
PSW.0	P	Parity flag, affected by hardware to indicate odd / even number of “one” bits in the Accumulator, i.e. even parity.

Bits RS1and RS0 are used to select the working register bank as follows:

Table 6. Register Bank selection

RS1:RS0	Bank selected	Location
00	Bank 0	(00h – 07h)
01	Bank 1	(08h – 0Fh)
10	Bank 2	(10h – 17h)
11	Bank 3	(18h – 1Fh)

Stack Pointer Register (SP)

The Stack Pointer is a 1-byte register initialized to 07h after reset. This register is incremented before PUSH and CALL instructions, causing the stack to begin at location 08h.

Data Pointer Register (DPL and DPH)

The Data Pointer (DPTR) is 2 bytes wide. The lower byte is DPL and the higher DPH. It can be loaded as either a single 2 byte register:

MOV DPTR,#data16)

or as two individual, single byte registers:

MOV DPL,#data8

MOV DPH,#data8

It is generally used to access external code or data space, for example:

MOVC A,@A+DPTR or

MOVX A,@DPTR.

Internal Program Memory Sizing Register (ROMSIZE)

The content of this register is used to determine the size of internal Program memory space. The addressable space is defined as:

ROMSIZE x 256

The register, which can have minimum and maximum values of 00h and FFh respectively, can therefore be used to define an internal ROM space, that is multiples of 256 bytes, in the range 0 – 64KB.

By default, the reset value of the ROMSIZE register is 10h, which gives an internal Program memory space of 4KB.

The register is loaded under software control.

Power Control Register (PCON)

Table 7. PCON register flags

MSB							LSB
SMOD	F6	F5	F4	F3	F2	0	0

Table 8. PCON register bit functions

Bit	Symbol	Function
PCON.7	SMOD	Double baud rate bit. If Timer 1 is used to generate the baud rate and SMOD is set (1), the baud rate is doubled when the Serial Port is used in modes 1,2 or 3
PCON.6	F6	General purpose Flag 6 available for user
PCON.5	F5	General purpose Flag 5 available for user
PCON.4	F4	General purpose Flag 4 available for user
PCON.3	F3	General purpose Flag 3 available for user
PCON.2	F2	General purpose Flag 2 available for user
PCON.1	0	This bit is read only and is permanently cleared (0)
PCON.0	0	This bit is read only and is permanently cleared (0)

Hardware Description

The TSK51x core is partitioned into modules as shown in figure 12 and described below.

Core Engine

The core engine of the TSK51x is composed of four components:

- Control Unit
- Arithmetic Logic Unit
- Memory Control Unit
- RAM and SFR Control Unit.

The TSK51x engine allows instructions to be fetched from Program memory and to execute using either RAM or SFR.

Arithmetic Logic Unit:

- 8 bit arithmetic operations
- 8 bit logical operations
- Boolean manipulations
- 8 x 8 bit multiplication
- 8 / 8 bit division

RAM and SFR Control Unit:

- Can address up to 256 bytes of Read/Write Data memory space
- Serves as Interface for off-core Special Function Registers

Memory Control Unit:

- Can address up to 64KB of internal Program memory space
- Can address up to 64KB of external Program memory space
- Can address up to 64KB of external Data memory space.

Block Diagram

Figure 5 shows the core engine and peripheral units for the TSK51x. Note that interface signals PSWR, ROMDATAO and ROMWR in the Memory Control Unit are present only in the debug-enabled version of the core – TSK51A_D.

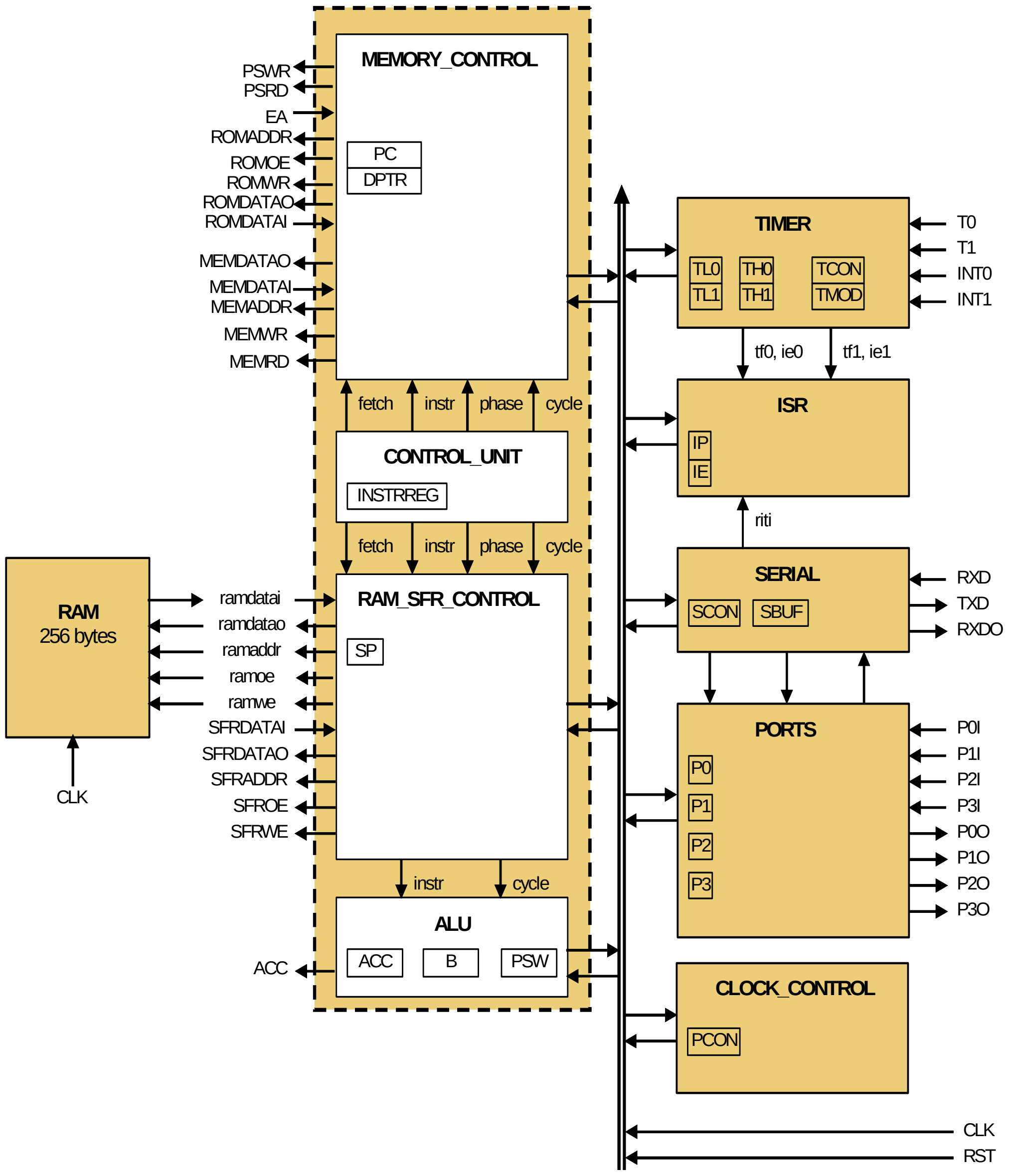


Figure 5. TSK51x Block diagram

Ports

Ports P0, P1, P2 and P3 are Special Function Registers. The contents of the SFR can be observed on the corresponding component symbol interface pins. Writing a ‘1’ to any of the ports causes the corresponding pin to be at the high level and writing a ‘0’ causes the corresponding pin to be held at the low level.

All four ports on the chip are bi-directional. Each of them consists of a Latch (SFR P0 to P3), an output drive and an input buffer, so the CPU can output or read data through any of these ports.

Timers / Counters

Timers 0 and 1

The TSK51x has two 16-bit timer/counter registers: Timer 0 and Timer 1. Both can be configured for counter or timer operations.

In timer mode, the register is incremented every machine (instruction) cycle, which means that it counts up after every 12 clock cycles.

In counter mode, the register is incremented when the falling edge is observed at the corresponding input pin T0 or T1. Since it takes two machine cycles to recognize a 1-to-0 event, the maximum input count rate is 1/24 of the external clock (CLK) frequency. There are no restrictions on the duty cycle, however to ensure proper recognition of 0 or 1 state, an input should be stable for at least one machine cycle (12 clock cycles).

Four operating modes can be selected for Timer 0 and Timer 1. Two Special Function Registers (TMOD and TCON) are used to select the appropriate mode.

Timer / Counter Mode Control Register(TMOD)

Table 9. The TMOD register flags

MSB							LSB
GATE	C/T	M1	M0	GATE	C/T	M1	M0
	Timer 1				*Timer 0*		

Table 10. The TMOD register bits description

Bit	Symbol	Function
TMOD.3 TMOD.7	GATE	When GATE = 0, Timer/Counter x will run only when TRx bit is set (see TCON register). This allows for Software Control. When GATE = 1, Timer/Counter x will run only when TRx bit is set (in TCON register) AND INTx pin is Low. This allows for Hardware Control.
TMOD.2 TMOD.6	C/T	When C/T = 0, Timer/Counter x will run as a timer, triggered by the internal clock. When C/T = 1, Timer/Counter x will run as a counter, triggered by the falling edge of the external signals entering pin T0 (for Timer/Counter 0) and T1 (for Timer/Counter 1).
TMOD.1 TMOD.5	M1	Selects mode for Timer/Counter 0 or Timer/Counter 1, as shown in Table 13.
TMOD.0 TMOD.4	M0	Selects mode for Timer/Counter 0 or Timer/Counter 1, as shown in Table 13.

Timer / Counter Control Register(TCON)

Table 11. The TCON register flags

MSB							LSB
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Table 12. The TCON register bit functions

Bit	Symbol	Function
TCON.7	TF1	Timer/Counter 1 overflow flag set by hardware when Timer/Counter 1 overflows. This flag is cleared by hardware.
TCON.6	TR1	Timer/Counter 1 Run control bit. If cleared, Timer/Counter 1 stops.
TCON.5	TF0	Timer/Counter 0 overflow flag set by hardware when Timer/Counter 0 overflows. This flag is cleared by hardware
TCON.4	TR0	Timer/Counter 0 Run control bit. If cleared, Timer/Counter 0 stops.
TCON.3	IE1	Interrupt 1 flag. Set by hardware when an interrupt of the type specified by IT1 is observed on external pin INT1. This flag is cleared when the interrupt is processed.
TCON.2	IT1	Interrupt 1 type control bit. Set/cleared by software to specify rising edge/high level triggered External Interrupt.
TCON.1	IE0	Interrupt 0 flag. Set by hardware when an interrupt of the type

		specified by IT0 is observed on external pin INT0. This flag is cleared when the interrupt is processed.
TCON.0	IT0	Interrupt 0 type control bit. Set/cleared by software to specify rising edge/high level triggered External Interrupt.

Timing Modes

Four modes of operation are supported for the two timers, determined by the state of bits M1 and M0 in the TMOD register (TMOD.1 and TMOD.0 respectively for Timer/Counter 0; TMOD.5 and TMOD.4 respectively for Timer/Counter 1). Table 13 summarizes the required states of these bits to achieve the desired operational mode.

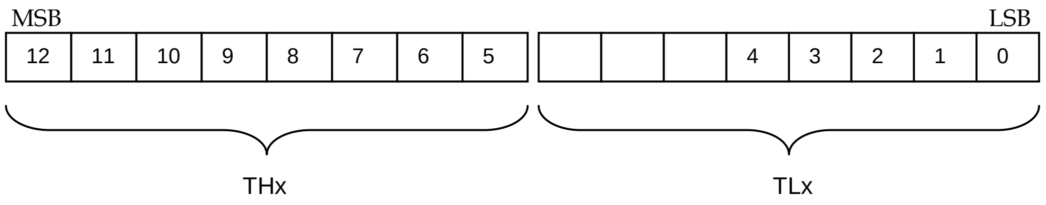
Table 13. Timer/Counter Modes

M1	M0	Mode
0	0	Mode 0
0	1	Mode 1
1	0	Mode 2
1	1	Mode 3

Mode 0

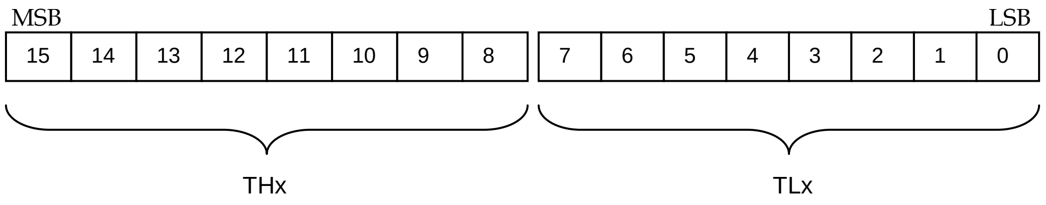
When in Mode 0, the Timer/Counter is set to 13 bits, where the 3 MSB bits of the TLx register are not used. Assuming the Timer/Counter is enabled, it will count from its set value (set by software) up to 1FFFh, at which point TFx is set to 1 to indicate overflow. Hardware then resets this value to 0.

At overflow the Timer/Counter rolls over to 0000h and continues to count up to 1FFFh, at which point TFx is set to 1 once again. This cycle continues until the Timer/Counter is disabled.



Mode 1

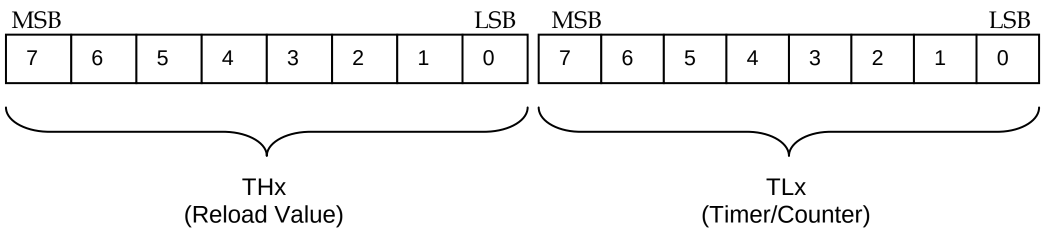
When in Mode 1, the Timer/Counter is set to 16 bits. The operation of the Timer/Counter in this mode is comparable to that in Mode 0. However, for this mode all 8 bits of the TLx register are used and therefore, the maximum value before overflow is FFFFh.



Mode 2

When in Mode 2, the Timer/Counter is set to 8 bits. This mode enables the Timer/Counter to be reloaded with its set value immediately after overflow. The two timing registers THx (the upper 8 bits) and TLx (the Lower 8 bits) are used differently.

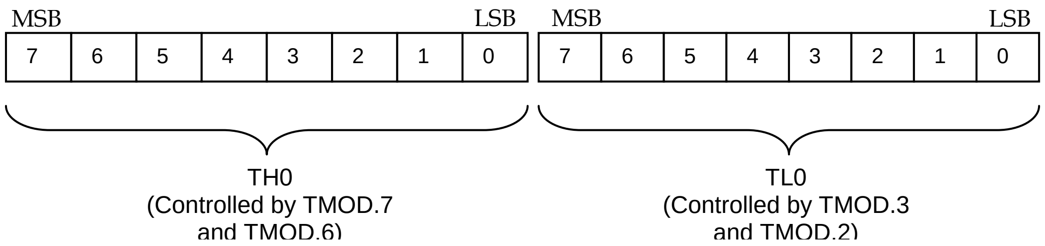
In this mode, THx holds the reload value, which is copied to TLx after overflow is detected, whereas TLx is the 8-bit dedicated Timer/Counter.



Mode 3

When Timer/Counter 1 is configured for operation in Mode 3, it is stopped. When Timer/Counter 0 is configured for operation in this same mode, the TH0 and TL0 registers operate independently of each other as follows:

- TL0 operates as an 8-bit Timer/Counter, controlled by Timer/Counter 0 mode control bits TMOD.3 and TMOD.2.
- TH0 operates as a dedicated 8-bit Timer, controlled by Timer/Counter 1 mode control bits TMOD.7 and TMOD.6, with no external gate control.



TL0, if enabled, will count from its set value to FFh, at which point the overflow flag for Timer/Counter 0 - TF0 (TCON.5) - is set to 1 and then reset to 0 by hardware. TL0 will continue to cycle through from 00h to FFh.

If Timer/Counter 1 is in Mode 3, then TH0, when enabled by Timer/Counter 1's mode control bits, will respond exactly the same as TL0, where TF1 (TCON.7) is set to 1 when overflow occurs. However, if Timer/Counter 1 is in Mode 0, 1 or 2, then the overflow flag, TF1, will be triggered by Timer/Counter 1 and TH0 of Timer/Counter 0.

Serial Interface

Serial Port 0

The serial buffer consists of two separate registers, a transmit buffer and a receive buffer. Writing data to the Special Function Register SBUF loads this data into the serial output buffer and starts the transmission. Reading from the SBUF register takes data from the serial receive buffer.

The serial port can simultaneously transmit and receive data. It can also buffer 1 byte at receive, which prevents the receive data from being lost if the CPU reads the first byte before transmission of the second byte is completed. The serial port can operate in 4 modes.

Mode 0

Pin RXD serves as input and RXDO as output. TXD outputs the shift clock. 8 bits are transmitted with LSB first. The baud rate is fixed at 1/12 of the external system clock frequency.

Mode 1

Pin RXD serves as input and TXD serves as serial output. No external shift clock is used. 10 bits are transmitted: a start bit (always 0), 8 data bits (LSB first), and a stop bit (always 1). On reception, the start bit synchronizes the transmission, 8 data bits are made available by reading SBUF, and the stop bit sets the flag RB8 in the Special Function Register SCON.

Mode 2

This mode is similar to Mode 1, with two differences. The baud rate is fixed at 1/32 or 1/64 of oscillator frequency, and 11 bits are transmitted or received: a start bit (0), 8 data bits (LSB first), a programmable 9th bit, and a stop bit (1). The 9th bit can be used to control the parity of the serial interface: at transmission, bit TB8 in SCON is output as the 9th bit, and at receive, the 9th bit affects RB8 in the Special Function Register SCON.

Mode 3

The only difference between Mode 2 and Mode 3 is that the baud rate is variable in Mode 3.

Reception is initialized in Mode 0 by setting the flags in SCON as follows: RI = 0 and REN = 1. In other modes, a start bit when REN = 1 starts receiving serial data

Multiprocessor Communication

The feature of receiving 9 bits in Modes 2 and 3 can be used for multiprocessor communication. In this case, the slave processors have bit SM2 in SCON set to 1. When the master processor outputs a slave's address, it sets the 9th bit to 1, causing a serial port receive interrupt in all the slaves. The slave processors compare the received byte with their network address. If there is a match, the addressed slave will clear SM2 and receive the rest of the message, while other slaves will leave the SM2 bit unaffected and ignore the message. After addressing the slave, the host will output the rest of the message with the 9th bit set to 0, so no serial port receive interrupt will be generated in unselected slaves.

Serial Port Control Register (SCON)

The function of the serial port depends on the status of the various flags in the Serial Port Control register SCON.

Table 14. The SCON register flags

MSB							LSB
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Table 15. The SCON register Bit functions

Bit	Symbol	Function
SCON.7	SM0	Sets baud rate
SCON.6	SM1	Sets baud rate
SCON.5	SM2	Enables multiprocessor communication feature.
SCON.4	REN	If set, enables serial reception. Cleared by software to disable reception.
SCON.3	TB8	The 9th transmitted data bit in Modes 2 and 3. Set or cleared by the CPU, depending on the function it performs (parity check, multiprocessor communication etc.)
SCON.2	RB8	In Modes 2 and 3, it is the 9th data bit received. In Mode 1, if SM2 is 0, RB8 is the stop bit. In Mode 0 this bit is not used. Must be cleared by software.
SCON.1	TI	Transmit interrupt flag, set by hardware after completion of a serial transfer. Must be cleared by software.
SCON.0	RI	Receive interrupt flag, set by hardware after completion of a serial reception. Must be cleared by software

Table 16. Serial Port Modes

SM0	SM1	Mode	Description	Baud Rate
0	0	0	Shift Register	F_{CLK} /12
0	1	1	8-bit UART	variable
1	0	2	9-bit UART	F_{CLK} /32 or /64
1	1	3	9-bit UART	variable

Table 17. Serial Port baud rates

Mode	Baud rate	
Mode 0	F_{CLK} / 12	
Mode 1,3	Timer 1 overflow rate	
Mode 2	SMOD = 0	F_{CLK} / 64
	SMOD = 1	F_{CLK} / 32

Note: SMOD is bit 7 in the Special Function Register PCON.

Generating Variable Baud Rate in Modes 1 and 3

In Modes 1 and 3, the Timer 1 overflow rate is used to generate baud rates. If Timer 1 is configured in auto – reload mode, to establish a baud rate the following equation is used:

$$\text{Baud Rate} = \frac{2^{SMOD} \times F_{CLK}}{32 \times 12 \times (256 - TH1)}$$

Reset

Hardware Reset (RST)

A reset is accomplished by holding the RST pin high for at least two instruction cycles (24 clock cycles) while the external clock (CLK) is running. The CPU responds by generating an internal reset, with the timing shown in Figure 6. The external reset signal is asynchronous to the internal clock. The RST pin is sampled on the rising edge, every 12 clock cycles.

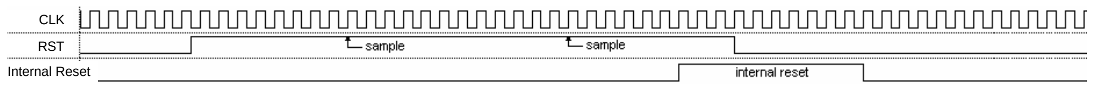


Figure 6. Reset timing

Note:	CLK	- clock oscillator input
	RST	- external reset input
	Internal Reset	- internal signal generated based on an external reset condition
	sample	- point at which the external reset input is sampled.

Reset Values

The internal reset signal is derived from the external reset (RST). It drives synchronous registers and flip-flops.

Table 18. Reset values

Register	Reset value
PC	0000h
ACC	00000000b
B	00000000b
XP	00000000b
PSW	00000000b
SP	00000000b
DPTR	0000h
P0	11111111b
P1	11111111b
P2	11111111b
P3	11111111b

Register	Reset value
IP	11100000b
IE	01100000b
TMOD	00000000b
TCON	00000000b
TH0	00000000b
TL0	00000000b
TH1	00000000b
TL1	00000000b
SCON	00000000b
SBUF	00000000b
ROMSIZE	00010000b
PCON	01111100b

Interrupts

The TSK51x provides five interrupt sources. There are two external interrupts accessible through pins INT0 and INT1, edge or level sensitive (rising edge or High level). There are, also, internal interrupts associated with Timer 0 and Timer 1 and an internal interrupt from the Serial Port.

External Interrupts

The choice between external interrupt level or transition activity is made by setting IT1 and IT0 bits in the Special Function Register TCON.

When the interrupt event happens, a corresponding Interrupt Control Bit is set (IE0 or IE1). This control bit triggers an interrupt if the appropriate interrupt bit is enabled.

When the interrupt service routine is vectored, the corresponding control bit (IE0 or IE1) is cleared provided that the edge triggered mode was selected. If level mode is active, the external requesting source controls flags IE0 or IE1 by the logic level on pins INT0 or INT1 (0 or 1).

Recognition of an interrupt event is possible if, during low to high transitions, both low and high levels last at least 1 machine (instruction) cycle (12 clock cycles).

Timer 0 and Timer 1 Interrupts

Timer 0 and 1 interrupts are generated by TF0 and TF1 flags, which are set by the rollover of Timer 0 and 1, respectively. When an interrupt is generated, the flag that caused this interrupt is cleared by the hardware, if the CPU accessed the corresponding interrupt service vector. This can be done only if this interrupt is enabled in the IE register.

Serial Port Interrupt

Serial Port interrupt is generated by logical OR of the flags TI and RI in the Special Function Register SCON. TI is set after completion of the transmit data. RI is set when the last bit of the incoming serial data was read. Neither RI nor TI is cleared by hardware, so the interrupt service routine must be responsible to clear these flags.

Interrupt Enable Register (IE)

Table 19. The IE register flags

MSB							LSB
EA	-	-	ES	ET1	EX1	ET0	EX0

Table 20. The IE register bit functions

Bit	Symbol	Function
IE.7	EA	If cleared, disables all interrupts. If set, bits 0 to 4 enable / disable interrupts.
IE.6	-	Not used
IE.5	-	Not used
IE.4	ES	If set, enables Serial Port interrupt. If cleared, the Serial Port interrupt is disabled.
IE.3	ET1	If set, enables Timer 1 overflow interrupt. If cleared, the Timer 1 interrupt is disabled.
IE.2	EX1	If set, enables external interrupt 1. If cleared, external interrupt 1 is disabled.
IE.1	ET0	If set, enables Timer 0 overflow interrupt. If cleared, the Timer 0 interrupt is disabled.
IE.0	EX0	If set, enables external interrupt 0. If cleared, external interrupt 0 is disabled.

Interrupt Priority Register (IP)

Table 21. The IP register flags

MSB							LSB
-	-	-	PS	PT1	PX1	PT0	PX0

Table 22. The IP register bit functions

Bit	Symbol	Function
IP.7	-	Not used
IP.6	-	Not used
IP.5	-	Not used
IP.4	PS	If set, defines high priority level for Serial Port interrupt. If cleared, Serial Port interrupt will be processed at low priority level.
IP.3	PT1	If set, defines high priority level for Timer 1 overflow interrupt. If cleared, the Timer 1 overflow interrupt will be processed at low priority level.
IP.2	PX1	If set, defines high priority level for external interrupt 1. If cleared, the external interrupt 1 will be processed at low priority level.
IP.1	PT0	If set, defines high priority level for Timer 0 overflow interrupt. If cleared, the Timer 0 overflow interrupt will be processed at low priority level.
IP.0	PX0	If set, defines high priority level for external interrupt 0. If cleared, the external interrupt 0 will be processed at low priority level.

Interrupt Priority Level Structure.

There are two priority levels in the TSK51x and any interrupt can be individually programmed to a high or low priority level. Modifying the appropriate bits in the Special Function Register IP can accomplish this. A low priority interrupt service routine will be interrupted by a high priority interrupt. However, the high priority interrupt can not be interrupted.

If two interrupts of the same priority level occur, an internal polling sequence determines which of them will be processed first. This polling sequence is a second priority structure defined as follows in Table 23.

Table 23. Interrupt Priority Level

Source	**Priority Within Level**
IE0	1 – highest
TF0	2
IE1	3
TF1	4
RI or TI	5 – lowest

Interrupt Handling

The interrupt flags are sampled during each machine cycle. The samples are polled during the next machine cycle. If an interrupt flag is captured, the interrupt system will generate an LCALL instruction to the appropriate service routine, provided that this is not disabled by the following conditions:

1. An interrupt of the same or higher priority is processed
2. The current machine cycle is not the last cycle of the instruction (the instruction can not be interrupted)
3. The instruction in progress is RETI or any write to IE or IP registers.

Note that if an interrupt is disabled and the interrupt flag is cleared before the blocking condition is removed, no interrupt will be generated since the polling cycle will not sample any active interrupt condition. In other words, the interrupt condition is not remembered. Every polling cycle is new.

On-Chip Debugging

The TSK51A_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 SFRs and PC
- Read-write access for Program memory and Data memory
- Unlimited software breakpoints
- User can specify whether the peripheral's clocks are stopped when processor enters debug mode.

Adding Debug Functionality to the Standard Core

The debug functionality of the TSK51A_D is provided through the use of an On-Chip Debug System unit (OCDS). The simplified block diagram of Figure 7 shows the connection between this unit and the standard TSK51A core.

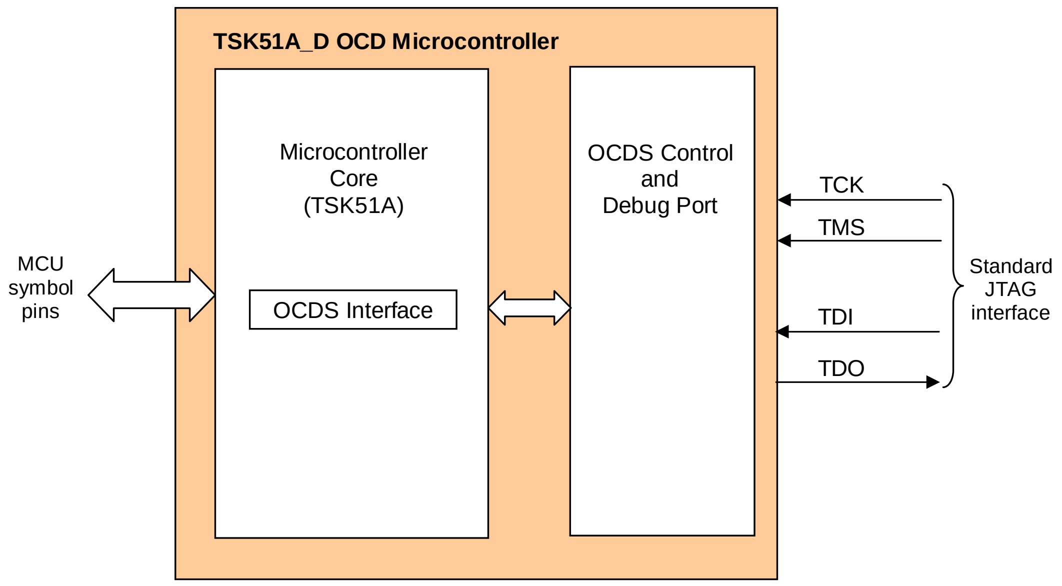

TSK51A_D OCD Microcontroller
Microcontroller Core (TSK51A)
OCDS Interface
OCDS Control and Debug Port
MCU symbol pins
TCK
TMS
TDI
TDO
Standard JTAG interface

Figure 7. Simplified TSK51A_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 8). 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.

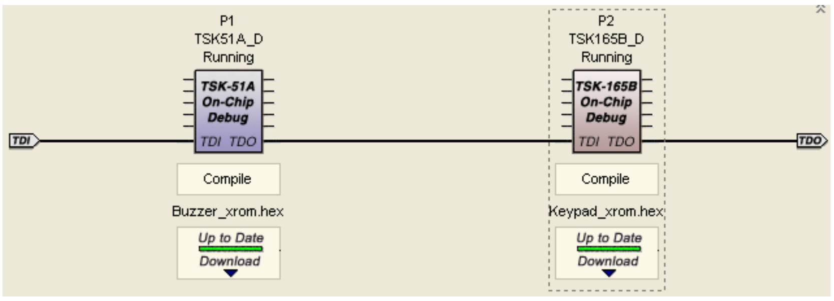


Figure 8. Nexus-enabled microcontrollers appearing in the Soft Devices chain

For microcontrollers such as the TSK51A_D, the Nexus protocol enables you to debug the core through communication with the OCDS Unit.

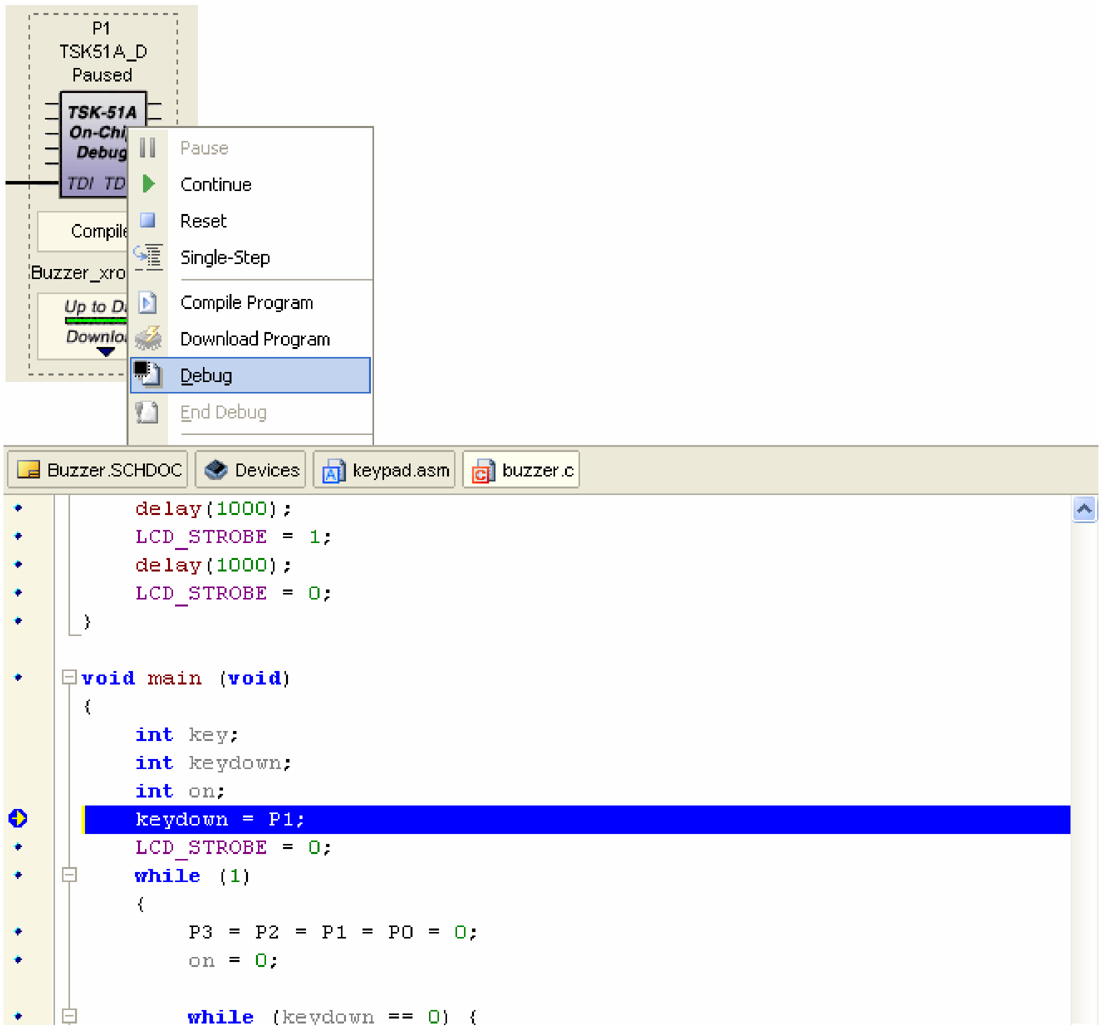
Accessing the Debug Environment

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 9).

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



```
        delay(1000);
        LCD_STROBE = 1;
        delay(1000);
        LCD_STROBE = 0;
    }

void main (void)
    {
        int key;
        int keydown;
        int on;
        keydown = P1;
        LCD_STROBE = 0;
        while (1)
        {
            P3 = P2 = P1 = P0 = 0;
            on = 0;

            while (keydown == 0) {
```

Figure 9. 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:

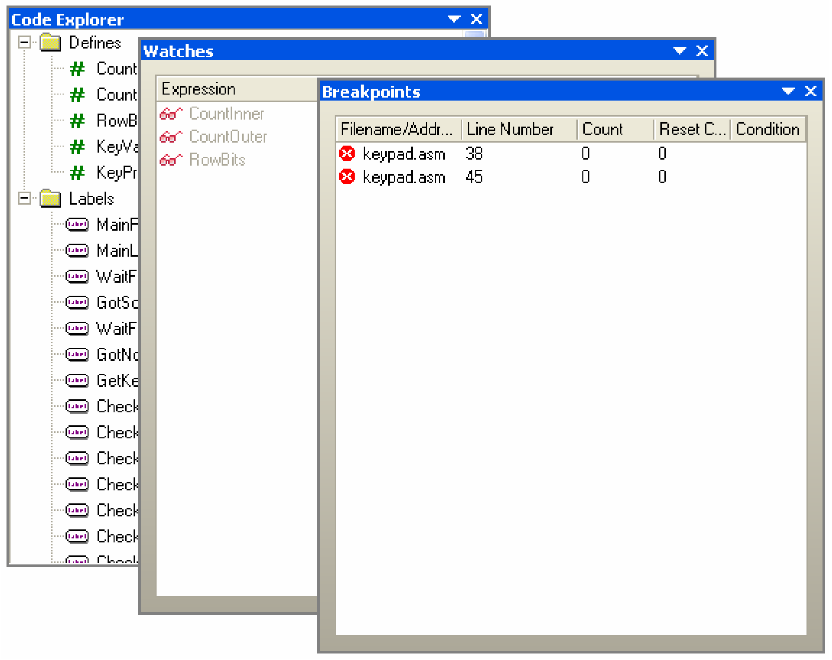
- Setting Breakpoints
- Adding Watches
- Stepping into and over at both the source (`*.C`) and instruction (`*.asm`) level
- Reset, Run and Halt code execution

- 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.



Code Explorer

```
Defines
  # Count
  # Count
  # RowB
  # KeyVa
  # KeyPr
Labels
  MainF
  MainL
  WaitF
  GotSc
  WaitF
  GotNc
  GetKe
  Check
  Check
  Check
  Check
  Check
  Check
```

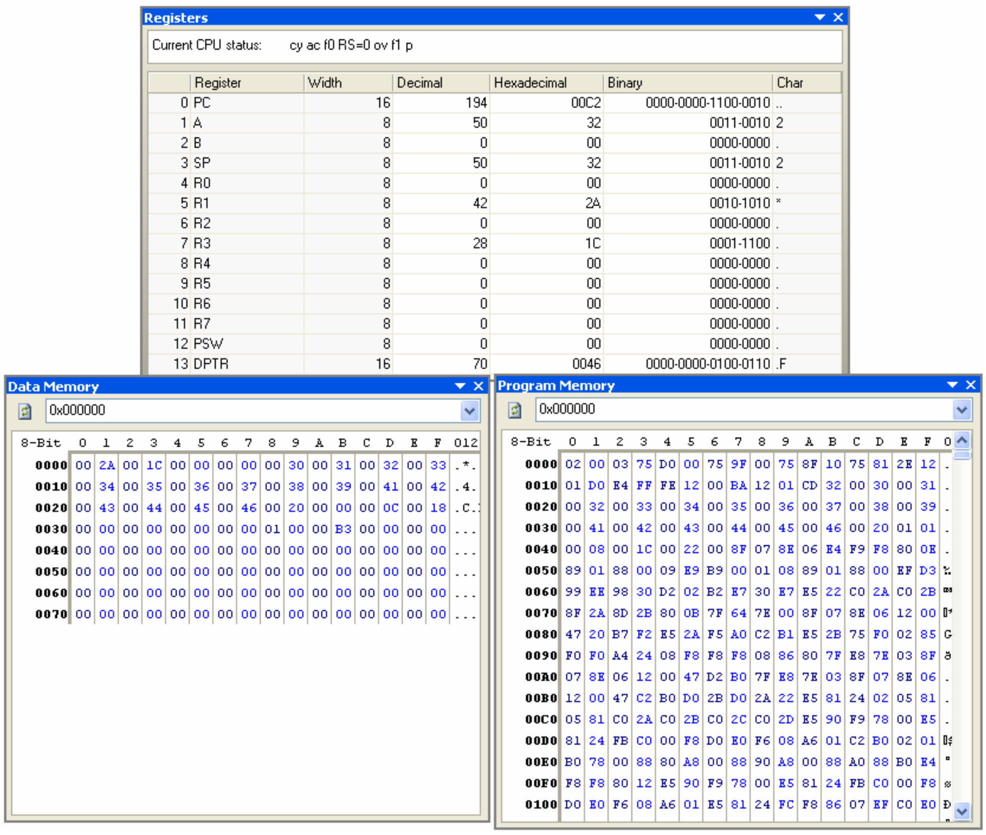
Watches

Expression
CountInner
CountOuter
RowBits

Breakpoints

Filename/Addr...	Line Number	Count	Reset C...	Condition
keypad.asm	38	0	0	
keypad.asm	45	0	0	

Figure 10. Workspace panels offering code-specific information and controls



Registers

Current CPU status: cy ac f0 RS=0 ov f1 p

	Register	Width	Decimal	Hexadecimal	Binary	Char
0	PC	16	194	00C2	0000-0000-1100-0010	..
1	A	8	50	32	0011-0010	2
2	B	8	0	00	0000-0000	.
3	SP	8	50	32	0011-0010	2
4	R0	8	0	00	0000-0000	.
5	R1	8	42	2A	0010-1010	*
6	R2	8	0	00	0000-0000	.
7	R3	8	28	1C	0001-1100	.
8	R4	8	0	00	0000-0000	.
9	R5	8	0	00	0000-0000	.
10	R6	8	0	00	0000-0000	.
11	R7	8	0	00	0000-0000	.
12	PSW	8	0	00	0000-0000	.
13	DPTR	16	70	0046	0000-0000-0100-0110	.F

Data Memory

0x000000

```
8-Bit  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  012
0000  00 2A 00 1C 00 00 00 00 00 30 00 31 00 32 00 33  .*.
0010  00 34 00 35 00 36 00 37 00 38 00 39 00 41 00 42  .4.
0020  00 43 00 44 00 45 00 46 00 20 00 00 00 0C 00 18  .C.
0030  00 00 00 00 00 00 00 00 01 00 00 B3 00 00 00 00  ...
0040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...
0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...
0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...
0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...
```

Program Memory

0x000000

```
8-Bit  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0
0000  02 00 03 75 D0 00 75 9F 00 75 8F 10 75 81 2E 12  .
0010  01 D0 E4 FF FE 12 00 BA 12 01 CD 32 00 30 00 31  .
0020  00 32 00 33 00 34 00 35 00 36 00 37 00 38 00 39  .
0030  00 41 00 42 00 43 00 44 00 45 00 46 00 20 01 01  .
0040  00 08 00 1C 00 22 00 8F 07 8E 06 E4 F9 F8 80 0E  .
0050  89 01 88 00 09 E9 B9 00 01 08 89 01 88 00 EF D3  %
0060  99 EE 98 30 D2 02 B2 E7 30 E7 E5 22 C0 2A C0 2B
0070  8F 2A 8D 2B 80 0B 7F 64 7E 00 8F 07 8E 06 12 00
0080  47 20 B7 F2 E5 2A F5 A0 C2 B1 E5 2B 75 F0 02 85  G
0090  F0 F0 A4 24 08 F8 F8 F8 08 86 80 7F E8 7E 03 8F  ð
00A0  07 8E 06 12 00 47 D2 B0 7F E8 7E 03 8F 07 8E 06  .
00B0  12 00 47 C2 B0 D0 2B D0 2A 22 E5 81 24 02 05 81  .
00C0  05 81 C0 2A C0 2B C0 2C C0 2D E5 90 F9 78 00 E5  .
00D0  81 24 FB C0 00 F8 D0 E0 F6 08 A6 01 C2 B0 02 01
00E0  B0 78 00 88 80 A8 00 88 90 A8 00 88 A0 88 B0 E4  °
00F0  F8 F8 80 12 E5 90 F9 78 00 E5 81 24 FB C0 00 F8  ø
0100  D0 E0 F6 08 A6 01 E5 81 24 FC F8 86 07 EF C0 E0  Ð
```

Figure 11. 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 access[3] 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 12).

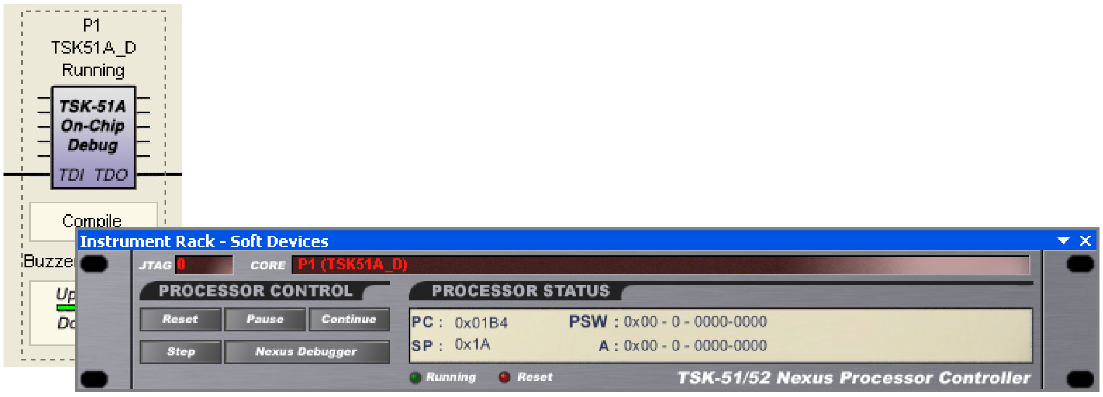


Figure 12. Accessing debug features from the microcontroller's instrument panel

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 13), 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.

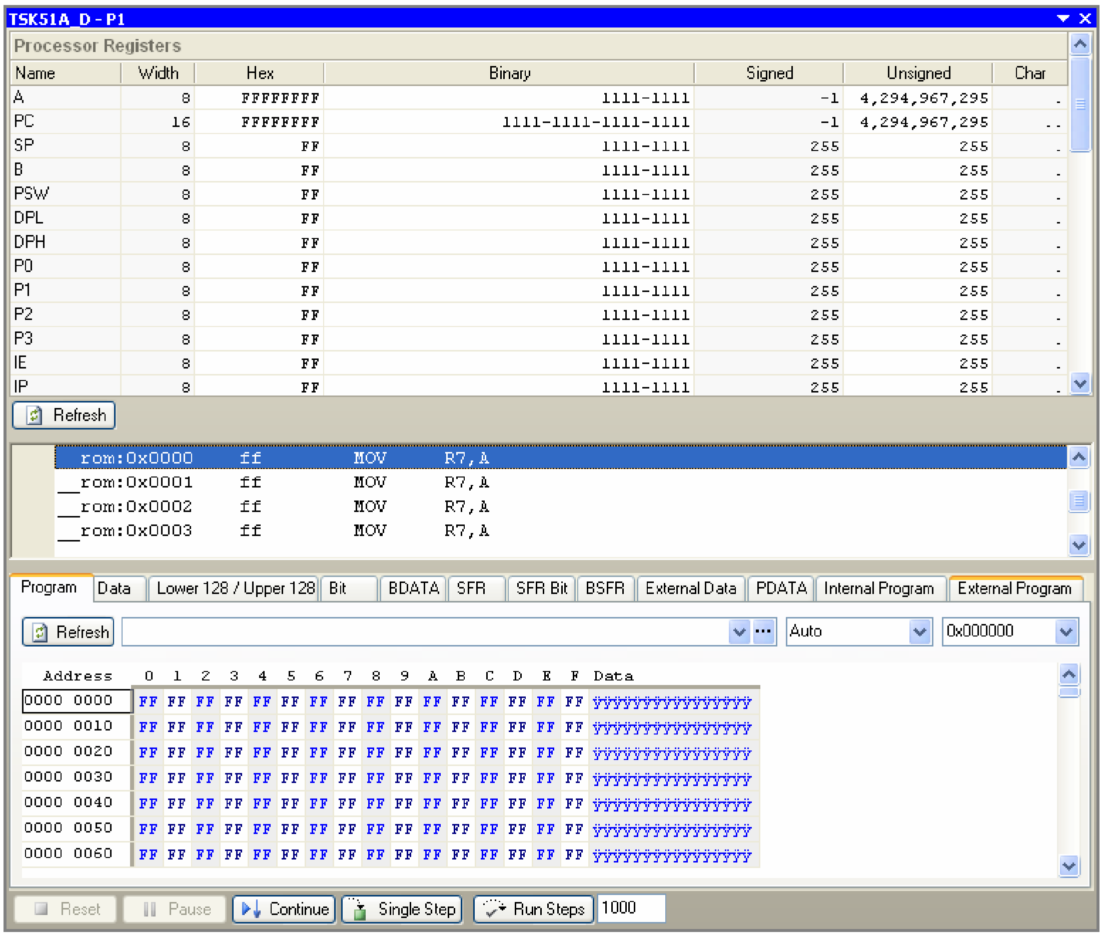
One key feature of the debug panel is that it enables you to specify (and therefore change) the embedded code (HEX file) that is downloaded to the microcontroller, quickly and efficiently.

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 TSK51x, see the *Using the TSK51x/TSK52x Embedded Tools* guide.

For comprehensive information with respect to the embedded tools available for the TSK51x, see the *TSK51x/TSK52x Embedded Tools Reference*.

[3] 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.



TSK51A_D - P1

Processor Registers

Name	Width	Hex	Binary	Signed	Unsigned	Char
A	8	FFFFFFFF	1111-1111	-1	4,294,967,295	.
PC	16	FFFFFFFF	1111-1111-1111-1111	-1	4,294,967,295	..
SP	8	FF	1111-1111	255	255	.
B	8	FF	1111-1111	255	255	.
PSW	8	FF	1111-1111	255	255	.
DPL	8	FF	1111-1111	255	255	.
DPH	8	FF	1111-1111	255	255	.
P0	8	FF	1111-1111	255	255	.
P1	8	FF	1111-1111	255	255	.
P2	8	FF	1111-1111	255	255	.
P3	8	FF	1111-1111	255	255	.
IE	8	FF	1111-1111	255	255	.
IP	8	FF	1111-1111	255	255	.

```
  rom:0x0000   ff   MOV   R7,A
__rom:0x0001   ff   MOV   R7,A
__rom:0x0002   ff   MOV   R7,A
__rom:0x0003   ff   MOV   R7,A
```

Auto 0x000000

```
Address    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  Data
0000 0000  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0000 0010  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0000 0020  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0000 0030  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0000 0040  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0000 0050  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0000 0060  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
```

Run Steps 1000

Figure 13. Processor debugging using the associated processor debug panel

Instruction Set

All TSK51x instructions are binary code compatible.

Table 24. Notes on data addressing modes

Rn	Working register R0-R7
direct	128 internal RAM locations, any I/O port, control or status register
@Ri	Indirect internal or external RAM location addressed by register R0 or R1
#data	8-bit constant included in instruction
#data16	16-bit constant included as bytes 2 and 3 of instruction
bit	128 software flags, any bit-addressable I/O pin, control or status bit
A	Accumulator

Table 25. Notes on program addressing modes

addr16	Destination address for LCALL and LJMP may be anywhere within the 64KB Program memory address space.
addr11	Destination address for ACALL and AJMP will be within the same 2KB page of Program memory as the first byte of the following instruction.
Rel	SJMP and all conditional jumps include an 8-bit offset byte. Range is +127/-128 bytes relative to the first byte of the following instruction.

Instruction Definitions

Table 26 shows the effect various instructions in the set have on the special function register PSW.

Only the carry, auxiliary carry and overflow flags are discussed. The parity bit is always computed from the actual content of the Accumulator.

Similarly, instructions which alter directly addressed registers could affect the other status flags if the instruction is applied to the PSW register. Status flags can also be modified by bit manipulation.

Table 26. PSW register flag modification (CY, OV, AC)

Instruction	Flag			Instruction	Flag		
	CY	OV	AC		CY	OV	AC
ADD	X	X	X	MOV C,bit	X		
ADDC	X	X	X	MUL	0	X	
ANL C,bit	X			ORL C,bit	X		
ANL C,/bit	X			ORL C,/bit	X		
CJNE	X			RLC	X		
CLR C	0			RRC	X		
CPL C	X			SETB C	1		
DA	X			SUBB	X	X	X
DIV	0	X					

Instruction Set - Functional Groupings

Table 27. Arithmetic operations

Mnemonic	Description	Width (in bytes)	No. of Instruction Cycles for execution
ADD A,#data	Add immediate data to Accumulator	2	1
ADD A,@Ri	Add indirect RAM to Accumulator	1	1
ADD A,direct	Add direct byte to Accumulator	2	1
ADD A,Rn	Add register to Accumulator	1	1
ADDC A,#data	Add immediate data to Accumulator with carry flag	2	1
ADDC A,@Ri	Add indirect RAM to Accumulator with carry flag	1	1
ADDC A,direct	Add direct byte to Accumulator with carry flag	2	1
ADDC A,Rn	Add register to Accumulator with carry flag	1	1
DA A	Decimal adjust Accumulator	1	1
DEC @Ri	Decrement indirect RAM	1	1
DEC A	Decrement Accumulator	1	1
DEC direct	Decrement direct byte	2	1
DEC Rn	Decrement register	1	1
DIV A,B	Divide Accumulator by B	1	4
INC @Ri	Increment indirect RAM	1	1
INC A	Increment Accumulator	1	1
INC direct	Increment direct byte	2	1
INC DPTR	Increment data pointer	1	2
INC Rn	Increment register	1	1
MUL A,B	Multiply Accumulator and B	1	4
SUBB A,#data	Subtract immediate data from Accumulator with borrow	2	1
SUBB A,@Ri	Subtract indirect RAM from Accumulator with borrow	1	1
SUBB A,direct	Subtract direct byte from Accumulator with borrow	2	1
SUBB A,Rn	Subtract register from Accumulator with borrow	1	1

Table 28. Logic operations

Mnemonic	**Description**	**Width (in bytes)**	**No. of Instruction Cycles for execution**
ANL A,#data	AND immediate data to Accumulator	2	1
ANL A,@Ri	AND indirect RAM to Accumulator	1	1
ANL A,direct	AND direct byte to Accumulator	2	1
ANL A,Rn	AND register to Accumulator	1	1
ANL direct,#data	AND immediate data to direct byte	3	2
ANL direct,A	AND Accumulator to direct byte	2	1
CLR A	Clear Accumulator	1	1
CPL A	Complement Accumulator	1	1
ORL A,#data	OR immediate data to Accumulator	2	1
ORL A,@Ri	OR indirect RAM to Accumulator	1	1
ORL A,direct	OR direct byte to Accumulator	2	1
ORL A,Rn	OR register to Accumulator	1	1
ORL direct,#data	OR immediate data to direct byte	3	2
ORL direct,A	OR Accumulator to direct byte	2	1
RL A	Rotate Accumulator left	1	1
RLC A	Rotate Accumulator left through carry	1	1
RR A	Rotate Accumulator right	1	1
RRC A	Rotate Accumulator right through carry	1	1
SWAP A	Swap nibbles within the Accumulator	1	1
XRL A,#data	Exclusive OR immediate data to Accumulator	2	1
XRL A,@Ri	Exclusive OR indirect RAM to Accumulator	1	1
XRL A,direct	Exclusive OR direct byte to Accumulator	2	1
XRL A,Rn	Exclusive OR register to Accumulator	1	1
XRL direct,#data	Exclusive OR immediate data to direct byte	3	2
XRL direct,A	Exclusive OR Accumulator to direct byte	2	1

Table 29. Data transfer

Mnemonic	Description	Width (in bytes)	No. of Instruction Cycles for execution
MOV @Ri, #data	Move immediate data to indirect RAM	2	1
MOV @Ri,A	Move Accumulator to indirect RAM	1	1
MOV @Ri,direct	Move direct byte to indirect RAM	2	2
MOV A,#data	Move immediate data to Accumulator	2	1
MOV A,@Ri	Move indirect RAM to Accumulator	1	1
MOV A,direct	Move direct byte to Accumulator	2	1
MOV A,Rn	Move register to Accumulator	1	1
MOV direct,#data	Move immediate data to direct byte	3	2
MOV direct,@Ri	Move indirect RAM to direct byte	2	2
MOV direct,A	Move Accumulator to direct byte	2	1
MOV direct,direct	Move direct byte to direct byte	3	2
MOV direct,Rn	Move register to direct byte	2	2
MOV DPTR, #data16	Load data pointer with a 16-bit constant	3	2
MOV Rn,#data	Move immediate data to register	2	1
MOV Rn,A	Move Accumulator to register	1	1
MOV Rn,direct	Move direct byte to register	2	2
MOVC A,@A + DPTR	Move code byte relative to DPTR to Accumulator	1	2
MOVC A,@A + PC	Move code byte relative to PC to Accumulator	1	2
MOVX @DPTR,A	Move Accumulator to external RAM (16-bit addr.)	1	2
MOVX @Ri,A	Move Accumulator to external RAM (8-bit addr.)	1	2
MOVX A,@DPTR	Move external RAM (16-bit addr.) to Accumulator	1	2
MOVX A,@Ri	Move external RAM (8-bit addr.) to Accumulator	1	2
POP direct	Pop direct byte from stack	2	2
PUSH direct	Push direct byte onto stack	2	2
XCH A,@Ri	Exchange indirect RAM with Accumulator	1	1
XCH A,direct	Exchange direct byte with Accumulator	2	1
XCH A,Rn	Exchange register with Accumulator	1	1
XCHD A,@Ri	Exchange low-order nibble of indirect RAM with Accumulator	1	1

Table 30. Boolean manipulation

Mnemonic	Description	Width (in bytes)	No. of Instruction Cycles for execution
ANL C, /bit	AND complement of direct bit to carry flag	2	2
ANL C,bit	AND direct bit to carry flag	2	2
CLR bit	Clear direct bit	2	1
CLR C	Clear carry flag	1	1
CPL bit	Complement direct bit	2	1
CPL C	Complement carry flag	1	1
MOV bit,C	Move carry flag to direct bit	2	2
MOV C,bit	Move direct bit to carry flag	2	1
ORL C, /bit	OR complement of direct bit to carry flag	2	2
ORL C,bit	OR direct bit to carry flag	2	2
SETB bit	Set direct bit	2	1
SETB C	Set carry flag	1	1

Table 31. Program branches

Mnemonic	Description	Width (in bytes)	No. of Instruction Cycles for execution
ACALL addr11	Absolute subroutine call	2	2
AJMP addr11	Absolute jump	2	2
CJNE @Ri,#data,rel	Compare immediate data to indirect RAM and jump if not equal	3	2
CJNE A,#data,rel	Compare immediate data to Accumulator and jump if not equal	3	2
CJNE A,direct,rel	Compare direct byte to Accumulator and jump if not equal	3	2
CJNE Rn,#data rel	Compare immediate data to register and jump if not equal	3	2
DJNZ direct,rel	Decrement direct byte and jump if not zero	3	2
DJNZ Rn,rel	Decrement register and jump if not zero	2	2
JB bit,rel	Jump if direct bit is set	3	2
JBC bit,rel	Jump if direct bit is set and clear bit	3	2
JC rel	Jump if carry flag is set	2	2
JMP @A + DPTR	Jump indirect relative to the DPTR	1	2
JNB bit,rel	Jump if direct bit is not set	3	2
JNC rel	Jump if carry flag is not set	2	2

Mnemonic	Description	Width (in bytes)	No. of Instruction Cycles for execution
JNZ rel	Jump if Accumulator is not zero	2	2
JZ rel	Jump if Accumulator is zero	2	2
LCALL addr16	Long subroutine call	3	2
LJMP addr16	Long jump	3	2
NOP	No operation	1	1
RET	Return from subroutine	1	2
RETI	Return from interrupt	1	2
SJMP rel	Short jump (relative addr.)	2	2

Hexadecimal Ordered Instructions

Table 32. Instruction Set in hexadecimal order

Opcode	Mnemonic	Opcode	Mnemonic
00h	NOP	10h	JBC bit,rel
01h	AJMP addr11	11h	ACALL addr11
02h	LJMP addr16	12h	LCALL addr16
03h	RR A	13h	RRC A
04h	INC A	14h	DEC A
05h	INC direct	15h	DEC direct
06h	INC @R0	16h	DEC @R0
07h	INC @R1	17h	DEC @R1
08h	INC R0	18h	DEC R0
09h	INC R1	19h	DEC R1
0Ah	INC R2	1Ah	DEC R2
0Bh	INC R3	1Bh	DEC R3
0Ch	INC R4	1Ch	DEC R4
0Dh	INC R5	1Dh	DEC R5
0Eh	INC R6	1Eh	DEC R6
0Fh	INC R7	1Fh	DEC R7
20h	JB bit.rel	30h	JNB bit.rel
21h	AJMP addr11	31h	ACALL addr11
22h	RET	32h	RETI
23h	RL A	33h	RLC A
24h	ADD A,#data	34h	ADDC A,#data
25h	ADD A,direct	35h	ADDC A,direct
26h	ADD A,@R0	36h	ADDC A,@R0
27h	ADD A,@R1	37h	ADDC A,@R1

Opcode	Mnemonic	Opcode	Mnemonic
28h	ADD A,R0	38h	ADDC A,R0
29h	ADD A,R1	39h	ADDC A,R1
2Ah	ADD A,R2	3Ah	ADDC A,R2
2Bh	ADD A,R3	3Bh	ADDC A,R3
2Ch	ADD A,R4	3Ch	ADDC A,R4
2Dh	ADD A,R5	3Dh	ADDC A,R5
2Eh	ADD A,R6	3Eh	ADDC A,R6
2Fh	ADD A,R7	3Fh	ADDC A,R7
40h	JC rel	50h	JNC rel
41h	AJMP addr11	51h	ACALL addr11
42h	ORL direct,A	52h	ANL direct,A
43h	ORL direct,#data	53h	ANL direct,#data
44h	ORL A,#data	54h	ANL A,#data
45h	ORL A,direct	55h	ANL A,direct
46h	ORL A,@R0	56h	ANL A,@R0
47h	ORL A,@R1	57h	ANL A,@R1
48h	ORL A,R0	58h	ANL A,R0
49h	ORL A,R1	59h	ANL A,R1
4Ah	ORL A,R2	5Ah	ANL A,R2
4Bh	ORL A,R3	5Bh	ANL A,R3
4Ch	ORL A,R4	5Ch	ANL A,R4
4Dh	ORL A,R5	5Dh	ANL A,R5
4Eh	ORL A,R6	5Eh	ANL A,R6
4Fh	ORL A,R7	5Fh	ANL A,R7
60h	JZ rel	70h	JNZ rel
61h	AJMP addr11	71h	ACALL addr11
62h	XRL direct,A	72h	ORL C,bit
63h	XRL direct,#data	73h	JMP @A+DPTR
64h	XRL A,#data	74h	MOV A,#data
65h	XRL A,direct	75h	MOV direct,#data
66h	XRL A,@R0	76h	MOV @R0,#data
67h	XRL A,@R1	77h	MOV @R1,#data
68h	XRL A,R0	78h	MOV R0.#data
69h	XRL A,R1	79h	MOV R1.#data
6Ah	XRL A,R2	7Ah	MOV R2.#data
6Bh	XRL A,R3	7Bh	MOV R3.#data

Opcode	Mnemonic	Opcode	Mnemonic
6Ch	XRL A,R4	7Ch	MOV R4.#data
6Dh	XRL A,R5	7Dh	MOV R5.#data
6Eh	XRL A,R6	7Eh	MOV R6.#data
6Fh	XRL A,R7	7Fh	MOV R7.#data
80h	SJMP rel	90h	MOV DPTR,#data16
81h	AJMP addr11	91h	ACALL addr11
82h	ANL C,bit	92h	MOV bit,C
83h	MOVC A,@A+PC	93h	MOVC A,@A+DPTR
84h	DIV AB	94h	SUBB A,#data
85h	MOV direct,direct	95h	SUBB A,direct
86h	MOV direct,@R0	96h	SUBB A,@R0
87h	MOV direct,@R1	97h	SUBB A,@R1
88h	MOV direct,R0	98h	SUBB A,R0
89h	MOV direct,R1	99h	SUBB A,R1
8Ah	MOV direct,R2	9Ah	SUBB A,R2
8Bh	MOV direct,R3	9Bh	SUBB A,R3
8Ch	MOV direct,R4	9Ch	SUBB A,R4
8Dh	MOV direct,R5	9Dh	SUBB A,R5
8Eh	MOV direct,R6	9Eh	SUBB A,R6
8Fh	MOV direct,R7	9Fh	SUBB A,R7
A0h	ORL C, /bit	B0h	ANL C, /bit
A1h	AJMP addr11	B1h	ACALL addr11
A2h	MOV C,bit	B2h	CPL bit
A3h	INC DPTR	B3h	CPL C
A4h	MUL AB	B4h	CJNE A,#data,rel
A5h	-	B5h	CJNE A,direct,rel
A6h	MOV @R0,direct	B6h	CJNE @R0,#data,rel
A7h	MOV @R1,direct	B7h	CJNE @R1,#data,rel
A8h	MOV R0,direct	B8h	CJNE R0,#data,rel
A9h	MOV R1,direct	B9h	CJNE R1,#data,rel
AAh	MOV R2,direct	BAh	CJNE R2,#data,rel
ABh	MOV R3,direct	BBh	CJNE R3,#data,rel
ACh	MOV R4,direct	BCh	CJNE R4,#data,rel
ADh	MOV R5,direct	BDh	CJNE R5,#data,rel
AEh	MOV R6,direct	BEh	CJNE R6,#data,rel
AFh	MOV R7,direct	BFh	CJNE R7,#data,rel

Opcode	Mnemonic	Opcode	Mnemonic
C0h	PUSH direct	D0h	POP direct
C1h	AJMP addr11	D1h	ACALL addr11
C2h	CLR bit	D2h	SETB bit
C3h	CLR C	D3h	SETB C
C4h	SWAP A	D4h	DA A
C5h	XCH A,direct	D5h	DJNZ direct,rel
C6h	XCH A,@R0	D6h	XCHD A,@R0
C7h	XCH A,@R1	D7h	XCHD A,@R1
C8h	XCH A,R0	D8h	DJNZ R0,rel
C9h	XCH A,R1	D9h	DJNZ R1,rel
CAh	XCH A,R2	DAh	DJNZ R2,rel
CBh	XCH A,R3	DBh	DJNZ R3,rel
CCh	XCH A,R4	DCh	DJNZ R4,rel
CDh	XCH A,R5	DDh	DJNZ R5,rel
CEh	XCH A,R6	DEh	DJNZ R6,rel
CFh	XCH A,R7	DFh	DJNZ R7,rel
E0h	MOVX A,@DPTR	F0h	MOVX @DPTR,A
E1h	AJMP addr11	F1h	ACALL addr11
E2h	MOVX A,@R0	F2h	MOVX @R0,A
E3h	MOVX A,@R1	F3h	MOVX @R1,A
E4h	CLR A	F4h	CPL A
E5h	MOV A,direct	F5h	MOV direct,A
E6h	MOV A,@R0	F6h	MOV @R0,A
E7h	MOV A,@R1	F7h	MOV @R1,A
E8h	MOV A,R0	F8h	MOV R0,A
E9h	MOV A,R1	F9h	MOV R1,A
EAh	MOV A,R2	FAh	MOV R2,A
EBh	MOV A,R3	FBh	MOV R3,A
ECh	MOV A,R4	FCh	MOV R4,A
EDh	MOV A,R5	FDh	MOV R5,A
EEh	MOV A,R6	FEh	MOV R6,A
EFh	MOV A,R7	FFh	MOV R7,A

Instruction Set – Detailed Reference

In the following detailed instruction set listing, @Ri is an indirect internal or external RAM location addressed by register R0 or R1. When this operand is used, the encoding for the instruction contains an entry 'I'. This will be replaced by a 0 or 1, depending on whether the register used is R0 or R1 respectively.

Similarly, the operand Rn can represent any of the eight working registers (R0-R7). The table below shows the registers that Rn can represent. The listed 3-bit value for each register replaces the rrr entry in the encoding for an instruction that uses this operand.

Register	rrr
R0	000
R1	001
R2	010
R3	011
R4	100
R5	101
R6	110
R7	111

ACALL addr11

Function: Absolute call

Description: ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the stack pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, op code bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of Program memory as the first byte of the instruction following ACALL. No flags are affected.

Operation: ACALL

(PC) ← (PC) + 2

(SP) ← (SP) + 1

((SP)) ← (PC7-0)

(SP) ← (SP) + 1

((SP)) ← (PC15-8)

(PC10-0) ← page address

Bytes: 2

Encoding:

a10	a9	a8	1	0	0	0	1	a7	a6	a5	a4	a3	a2	a1	a0

ADD A, <src-byte>

Function: Add

Description: ADD adds the byte variable indicated to the accumulator, leaving the result in the Accumulator. The carry and auxiliary carry flags are set, respectively, if there is a carry out of bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred. OV is set if there is a carry out of bit 6 but not out of bit 7, or a carry out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands. Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

ADD A, Rn

Operation: ADD

(A) ← (A) + (Rn)

Bytes: 1

Encoding:

0	0	1	0	1	r	r	r

ADD A, direct

Operation: ADD

(A) ← (A) + (direct)

Bytes: 2

Encoding:

0	0	1	0	0	1	0	1	direct address

ADD A, @Ri

Operation: ADD

(A) ← (A) + ((Ri))

Bytes: 1

Encoding:

0	0	1	0	0	1	1	i

ADD A, #data

Operation: ADD

(A) ← (A) + #data

Bytes: 2

Encoding:

0	0	1	0	0	1	0	0	immediate data

ADDC A, < src-byte>

Function: Add with carry

Description: ADDC simultaneously adds the byte variable indicated, the carry flag and the Accumulator contents, leaving the result in the accumulator. The carry and auxiliary carry flags are set, respectively, if there is a carry out of bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred. OV is set if there is a carry out of bit 6 but not out of bit 7, or a carry out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands. Four source operand-addressing modes are allowed: register, direct, register- indirect, or immediate.

ADDC A, Rn

Operation: ADDC

(A) ← (A) + (C) + (Rn)

Bytes: 1

Encoding:

0	0	1	1	1	r	r	r

ADDC A, direct

Operation: ADDC

(A) ← (A) + (C) + (direct)

Bytes: 2

Encoding:

0	0	1	1	0	1	0	1	direct address

ADDC A, @Ri

Operation: ADDC

(A) ← (A) + (C) + ((Ri))

Bytes: 1

Encoding:

0	0	1	1	0	1	1	i

ADDC A, #data

Operation: ADDC

(A) ← (A) + (C) + #data

Bytes: 2

Encoding:

0	0	1	1	0	1	0	0	immediate data

AJMP addr11

Function: Absolute jump

Description: AJMP transfers program execution to the indicated address, which is formed at run- time by concatenating the high-order five bits of the PC (*after* incrementing the PC twice), op code bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of Program memory as the first byte of the instruction following AJMP.

Operation: AJM P

(PC) ← (PC) + 2

(PC10-0) ← page address

Bytes: 2

Encoding:

a10	a9	a8	0	0	0	0	1	a7	a6	a5	a4	a3	a2	a1	a0

ANL <dest-byte>, <src-byte>

Function: Logical AND for byte variables

Description: ANL performs the bit wise logical AND operation between the variables indicated and stores the results in the destination variable. No flags are affected (except P, if <dest-byte> = A). The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

ANL A,Rn

Operation: ANL

(A) ← (A) ^ (Rn)

Bytes: 1

Encoding:

0	1	0	1	1	r	r	r

ANL A,direct

Operation: ANL

(A) ← (A) ^ (direct)

Bytes: 2

Encoding:

0	1	0	1	0	1	0	1	direct address

ANL A, @Ri

Operation: ANL

(A) ← (A) ^ ((Ri))

Bytes: 1

Encoding:

0	1	0	1	0	1	1	i

ANL A, #data

Operation: ANL

(A) ← (A) ^ #data

Bytes: 2

Encoding:

0	1	0	1	0	1	0	0	immediate data

ANL direct,A

Operation: ANL

(direct) ← (direct) ^ (A)

Bytes: 2

Encoding:

0	1	0	1	0	0	1	0	direct address

ANL direct, #data

Operation: ANL

(direct) ← (direct) ^ #data

Bytes: 3

Encoding:

0	1	0	1	0	0	1	1
direct address							
immediate data							

ANL C, <src-bit>

Function: Logical AND for bit variables

Description: If the Boolean value of the source bit is a logic 0 then clear the carry flag; otherwise leave the carry flag in its current state. (A slash "/" preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affected)*. No other flags are affected. Only direct bit addressing is allowed for the source operand.

ANL C,bit

Operation: ANL

(C) ← (C) ^ (bit)

Bytes: 2

Encoding:

1	0	0	0	0	0	1	0	bit address

ANL C,/bit

Operation: ANL

(C) ← (C) ^ / (bit)

Bytes: 2

Encoding:

1	0	1	1	0	0	0	0	bit address

CJNE <dest-byte >, < src-byte >, rel

Function: Compare and jump if not equal

Description: CJNE compares the magnitudes of the first two operands and branches if their values are not equal. The branch destination is computed by adding the signed relative displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected. The first two operands allow four addressing mode combinations: the Accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

CJNE A,direct,rel

Operation: CJNE

(PC) ← (PC) + 3

if (A) < > (direct)

then (PC) ← (PC) + relative offset

if (A) < (direct)

then (C) ←1

else (C) ←0

Bytes: 3

Encoding:

1	0	1	1	0	1	0	1
direct address							
relative address							

CJNE A, #data,rel

Operation: CJNE

(PC) ← (PC) + 3
if (A) < > data
then (PC) ← (PC) + relative offset
if (A) < data
then (C) ←1
else (C) ← 0

Bytes: 3

Encoding:

1	0	1	1	0	1	0	0
immediate data							
relative address							

CJNE Rn, #data, rel

Operation: CJNE

(PC) ← (PC) + 3
if (Rn) < > data
then (PC) ← (PC) + relative offset
if (Rn) < data
then (C) ← 1
else (C) ← 0

Bytes: 3

Encoding:

1	0	1	1	1	r	r	r
immediate data							
relative address							

CJNE @Ri, #data, rel

Operation: CJNE

(PC) ← (PC) + 3
if ((Ri)) < > data
then (PC) ← (PC) + relative offset
if ((Ri)) < data
then (C) ← 1
else (C) ← 0

Bytes: 3

Encoding:

1	0	1	1	0	1	1	i
immediate data							
relative address							

CLR A

Function: Clear Accumulator

Description: The Accumulator is cleared (all bits set to zero). No flags are affected.

Operation: CLR

(A) ← 0

Bytes: 1

Encoding:

1	1	1	0	0	1	0	0

CLR bit

Function: Clear bit

Description: The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on any directly addressable bit.

Operation: CLR

(bit) ← 0

Bytes: 2

Encoding:

1	1	0	0	0	0	1	0	bit address

CLR C

Function: Clear carry flag

Description: The carry flag is cleared (reset to zero). No other flags are affected.

Operation: CLR

(C) ← 0

Bytes: 1

Encoding:

1	1	0	0	0	0	1	1

CPL A

Function: Complement Accumulator

Description: Each bit of the Accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to zero and vice versa. No flags are affected.

Operation: CPL

(A) ← / (A)

Bytes: 1

Encoding:

1	1	1	1	0	1	0	0

CPL bit

Function: Complement bit

Description: The bit variable specified is complemented. A bit which had been a one is changed to zero and vice versa. No other flags are affected. CPL can operate on any directly addressable bit.

Note: When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, not the input pin.

Operation: CPL

(bit) ← / (bit)

Bytes: 2

Encoding:

1	0	1	1	0	0	1	0	bit address

CPL C

Function: Complement carry flag

Description: The carry flag is complemented. If the flag had been a one, it is changed to zero and vice versa. No other flags are affected.

Operation: CPL

$(C) \leftarrow / (C)$

Bytes: 1

Encoding:

1	0	1	1	0	0	1	1

DA A

Function: Decimal adjust Accumulator for addition

Description: DA A adjusts the eight-bit value in the Accumulator resulting from the earlier addition of two variables (each in packed BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition. If Accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the Accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set, or if the four high-order bits now exceed nine (1010xxxx-1111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. This instruction performs the decimal conversion by simply adding 00h , 06h , 60h , or 66h to the Accumulator, depending on initial Accumulator and PSW register conditions.

Note: DA A *cannot* simply convert a hexadecimal number in the Accumulator to BCD notation, nor does DA A apply to decimal subtraction.

Operation: DA

contents of Accumulator are BCD

if $[[(A_{3\text{-}0}) > 9] \vee [(AC) = 1]]$

then $(A_{3\text{-}0}) \leftarrow (A_{3\text{-}0}) + 6$

and

if $[[(A_{7\text{-}4}) > 9] \vee [(C) = 1]]$

then $(A_{7\text{-}4}) \leftarrow (A_{7\text{-}4}) + 6$

Bytes: 1

Encoding:

1	1	0	1	0	1	0	0

DEC byte

Function: Decrement

Description: The variable indicated is decremented by 1. An original value of 00h will underflow to 0FFh. No flags are affected. Four operand addressing modes are allowed: Accumulator, register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

DEC A

Operation: DEC

(A) ← (A) - 1

Bytes: 1

Encoding:

0	0	0	1	0	1	0	0

DEC Rn

Operation: DEC

(Rn) ← (Rn) - 1

Bytes: 1

Encoding:

0	0	0	1	1	r	r	r

DEC direct

Operation: DEC

(direct) ← (direct) - 1

Bytes: 2

Encoding:

0	0	0	1	0	1	0	1	direct address

DEC @Ri

Operation: DEC

((Ri)) ← ((Ri)) - 1

Bytes: 1

Encoding:

0	0	0	1	0	1	1	i

DIV AB

Function: Divide

Description: DIV AB divides the unsigned eight-bit integer in the Accumulator by the unsigned eight-bit integer in register B. The Accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared.

Exception: If B had originally contained 00h, the values returned in the Accumulator and B register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

Operation: DIV

(A) ← 15-8

(A) / (B)

(B) ← 7-0

Bytes: 1

Encoding:

1	0	0	0	0	1	0	0

DJNZ <byte>, <rel-addr>

Function: Decrement and jump if not zero

Description: DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00h will underflow to 0FFh. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction. The location decremented may be a register or directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

DJNZ Rn,rel

Operation: DJNZ

(PC) ← (PC) + 2

(Rn) ← (Rn) - 1

if (Rn) > 0 or (Rn) < 0

then (PC) ← (PC) + rel

Bytes: 2

Encoding:

1	1	0	1	1	r	r	r	rel. address

DJNZ direct,rel

Operation: DJNZ

(PC) ← (PC) + 2

(direct) ← (direct) - 1

if (direct) > 0 or (direct) < 0

then (PC) ← (PC) + rel

Bytes: 3

Encoding:

1	1	0	1	0	1	0	1
direct address							
relative address							

INC <byte>

Function: Increment

Description: INC increments the indicated variable by 1. An original value of 0FFh will overflow to 00h. No flags are affected. Four operand addressing modes are allowed: Accumulator, register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

INC A

Operation: INC

(A) ← (A) + 1

Bytes: 1

Encoding:

0	0	0	0	0	1	0	0

INC Rn

Operation: INC

(Rn) ← (Rn) + 1

Bytes: 1

Encoding:

0	0	0	0	1	r	r	r

INC direct

Operation: INC

(direct) ← (direct) + 1

Bytes: 2

Encoding:

0	0	0	0	0	1	0	1	direct address

INC @Ri

Operation: INC

((Ri)) ← ((Ri)) + 1

Bytes: 1

Encoding:

0	0	0	0	0	1	1	i

INC DPTR

Function: Increment data pointer

Description: Increment the 16-bit data pointer by 1. A 16-bit increment (modulo 2^{16}) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFh to 00h will increment the high-order byte (DPH). No flags are affected. This is the only 16-bit register which can be incremented.

Operation: INC

(DPTR) ← (DPTR) + 1

Bytes: 1

Encoding:

1	0	1	0	0	0	1	1

JB bit, rel

Function: Jump if bit is set

Description: If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. The bit tested is not modified. No flags are affected.

Operation: JB

(PC) ← (PC) + 3

if (bit) = 1

then (PC) ← (PC) + rel

Bytes: 3

Encoding:

0	0	1	0	0	0	0	0
bit address							
relative address							

JBC bit,rel

Function: Jump if bit is set and clear bit

Description: If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. *In either case, clear the designated bit.* The branch destination is computed by adding the signed relative displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

Note: When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, not the input pin.

Operation:

```
JBC
(PC) ← (PC) + 3
if (bit) = 1
then (bit) ← 0
(PC) ← (PC) + rel
```

Bytes: 3

Encoding:

0	0	0	1	0	0	0	0
bit address							
relative address							

JC rel

Function: Jump if carry flag is set

Description: If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative- displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

Operation:

```
JC
(PC) ← (PC) + 2
if (C) = 1
then (PC) ← (PC) + rel
```

Bytes: 2

Encoding:

0	1	0	0	0	0	0	0	relative address

JMP @A + DPTR

Function: Jump indirect relative to DPTR

Description: Add the eight-bit unsigned contents of the Accumulator with the 16-bit data pointer (DPTR), and load the resulting sum into the Program Counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2^{16}): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the Accumulator nor the data pointer is altered. No flags are affected.

Operation:

```
JMP
(PC) ← (A) + (DPTR)
```

Bytes: 1

Encoding:

0	1	1	1	0	0	1	1

JNB bit,rel

Function:	Jump if bit is not set
Description:	If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.
Operation:	JNB (PC) ← (PC) + 3 if (bit) = 0 then (PC) ← (PC) + rel.
Bytes:	3
Encoding:	

0	0	1	1	0	0	0	0
bit address							
relative address							

JNC rel

Function:	Jump if carry flag is not set
Description:	If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.
Operation:	JNC (PC) ← (PC) + 2 if (C) = 0 then (PC) ← (PC) + rel
Bytes:	2
Encoding:	

0	1	0	1	0	0	0	0	relative address

JNZ rel

Function:	Jump if Accumulator is not zero
Description:	If any bit of the Accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.
Operation:	JNZ (PC) ← (PC) + 2 if (A) $\neq$ 0 then (PC) ← (PC) + rel.
Bytes:	2
Encoding:	

0	1	1	1	0	0	0	0	relative address

JZ rel

Function:	Jump if Accumulator is zero
Description:	If all bits of the Accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.
Operation:	JZ (PC) ← (PC) + 2 if (A) = 0 then (PC) ← (PC) + rel
Bytes:	2
Encoding:	

0	1	1	0	0	0	0	0	relative address

LCALL addr16

Function:	Long call
Description:	LCALL calls a subroutine located at the indicated address. The instruction adds three to the Program Counter to generate the address of the next instruction and then pushes the 16-bit result onto the Stack (low byte first), incrementing the Stack Pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64KB Program memory address space. No flags are affected.
Operation:	LCALL (PC) ← (PC) + 3 (SP) ← (SP) + 1 ((SP)) ← (PC_{7-0}) (SP) ← (SP) + 1 ((SP)) ← (PC_{15-8}) (PC) ← addr15-0
Bytes:	3
Encoding:	

0	0	0	1	0	0	1	0
addr15-8							
addr7-0							

LJMP addr16

Function:	Long jump
Description:	LJMP causes an unconditional branch to the indicated address, by loading the high- order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64KB Program memory address space. No flags are affected.
Operation:	LJMP (PC) ← addr15... addr0
Bytes:	3

Encoding:

0	0	0	0	0	0	1	0
addr15-8							
addr7-0							

MOV <dest-byte>, <src-byte>

Function: Move byte variable

Description: The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected. This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

MOV A,Rn

Operation: MOV

(A) ← (Rn)

Bytes: 1

Encoding:

1	1	1	0	1	r	r	r

MOV A,direct

Operation: MOV

(A) ← (direct)

Bytes: 2

Note: MOV A,ACC is not a valid instruction. The content of the Accumulator after the execution of this instruction is undefined.

Encoding:

1	1	1	0	0	1	0	1	direct address

MOV A,@Ri

Operation: MOV

(A) ← ((Ri))

Bytes: 1

Encoding:

1	1	1	0	0	1	1	i

MOV A, #data

Operation: MOV

(A) ← #data

Bytes: 2

Encoding:

0	1	1	1	0	1	0	0	immediate data

MOV Rn,A

Operation: MOV

(Rn) ← (A)

Bytes: 1

Encoding:

1	1	1	1	1	r	r	r

MOV Rn,direct

Operation: MOV

(Rn) ← (direct)

Bytes: 2

Encoding:

1	0	1	0	1	r	r	r	direct address

MOV Rn, #data

Operation: MOV

(Rn) ← #data

Bytes: 2

Encoding:

0	1	1	1	1	r	r	r	immediate data

MOV direct,A

Operation: MOV

(direct) ← (A)

Bytes: 2

Encoding:

1	1	1	1	0	1	0	1	direct address

MOV direct,Rn

Operation: MOV

(direct) ← (Rn)

Bytes: 2

Encoding:

1	0	0	0	1	r	r	r	direct address

MOV direct,direct

Operation: MOV

(direct) ← (direct)

Bytes: 3

Encoding:

1	0	0	0	0	1	0	1
Direct address (source)							
Direct address (destination)							

MOV direct, @ Ri

Operation: MOV

(direct) ← ((Ri))

Bytes: 2

Encoding:

1	0	0	0	0	1	1	i	direct address

MOV direct, #data

Operation: MOV

(direct) ← #data

Bytes: 3

Encoding:

0	1	1	1	0	1	0	1
direct address							
immediate data							

MOV @ Ri,A

Operation: MOV

((Ri)) ← (A)

Bytes: 1

Encoding:

1	1	1	1	0	1	1	i

MOV @ Ri,direct

Operation: MOV

((Ri)) ← (direct)

Bytes: 2

Encoding:

1	0	1	0	0	1	1	i	direct address

MOV @ Ri,#data

Operation: MOV

((Ri)) ← #data

Bytes: 2

Encoding:

0	1	1	1	0	1	1	i	immediate data

MOV <dest-bit>, <src-bit>

Function: Move bit data

Description: The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.

MOV C,bit

Operation: MOV

(C) ← (bit)

Bytes: 2

Encoding:

1	0	1	0	0	0	1	0	bit address

MOV bit,C

Operation: MOV

(bit) ← (C)

Bytes: 2

Encoding:

1	0	0	1	0	0	1	0	bit address

MOV DPTR, #data16

Function: Load data pointer with a 16-bit constant

Description: The data pointer is loaded with the 16-bit constant indicated. The 16 bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.

This is the only instruction which moves 16 bits of data at once.

Operation: MOV

(DPTR) ← #data15..0

DPH DPL ← #data15...8 #data7..0

Bytes: 3

Encoding:

1	0	0	1	0	0	0	0
immediate data 15-8							
immediate data 7-0							

MOVC A, @A + <base-reg>

Function: Move code byte

Description The MOVC instructions load the Accumulator with a code byte, or constant from Program memory. The address of the byte fetched is the sum of the original unsigned eight-bit Accumulator contents and the contents of a sixteen-bit base register, which may be either the data pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added to the Accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

MOVC A, @A + DPTR

Operation: MOVC

(A) ← ((A) + (DPTR))

Bytes: 1

Encoding:

1	0	0	1	0	0	1	1

MOVC A, @A + PC

Operation: MOVC

(PC) ← (PC) + 1

(A) ← ((A) + (PC))

Bytes: 1

Encoding:

1	0	0	0	0	0	1	1

MOVX <dest-byte>, <src-byte>

Function:	Move external
Description:	The MOVX instructions transfer data between the Accumulator and a byte of external Data memory, hence the X appended to MOV. There are two types of instructions, differing in whether they provide an 8-bit or 16-bit indirect address to the external Data RAM. In the first type, the contents of R0 or R1 in the current register bank provide an 8-bit address. In the second type, the data pointer generates a 16-bit address.

MOVX A,@Ri

Operation: MOVX

$(A) \leftarrow ((Ri))$

Bytes: 1

Encoding:

1	1	1	0	0	0	1	i

MOVX A,@DPTR

Operation: MOVX

$(A) \leftarrow ((DPTR))$

Bytes: 1

Encoding:

1	1	1	0	0	0	0	0

MOVX @Ri,A

Operation: MOVX

$((Ri)) \leftarrow (A)$

Bytes: 1

Encoding:

1	1	1	1	0	0	1	i

MOVX @DPTR,A

Operation: MOVX

$((DPTR)) \leftarrow (A)$

Bytes: 1

Encoding:

1	1	1	1	0	0	0	0

MUL AB

Function: Multiply

Description: MUL AB multiplies the unsigned 8-bit integers in the Accumulator and register B. The low-order byte of the 16-bit product is left in the Accumulator and the high-order byte in register B. If the product is greater than 255 (0FFh) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

Operation: MUL

$(A) \leftarrow_{7-0}$

$(A) \times (B)$

$(B) \leftarrow_{15-8}$

Bytes: 1

Encoding:

1	0	1	0	0	1	0	0

NOP

Function: No operation

Description: Execution continues at the following instruction. Other than the PC, no registers or flags are affected.

Operation: NOP

(PC) ← (PC) + 1)

Bytes: 1

Encoding:

0	0	0	0	0	0	0	0

ORL <dest-byte>, <src-byte>

Function: Logical OR for byte variables

Description: ORL performs the bit wise logical OR operation between the indicated variables, storing the results in the destination byte. No flags are affected (except P (parity bit), if <dest-byte> = A).

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be either the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

ORL A,Rn

Operation: ORL

(A) ← (A) ∨ (Rn)

Bytes: 1

Encoding:

0	1	0	0	1	r	r	r

ORL A,direct

Operation: ORL

(A) ← (A) ∨ (direct)

Bytes: 2

Encoding:

0	1	0	0	0	1	0	1	direct address

ORL A,@Ri

Operation: ORL

(A) ← (A) ∨ ((Ri))

Bytes: 1

Encoding:

0	1	0	0	0	1	1	i

ORL A,#data

Operation: ORL

(A) ← (A) ∨ #data

Bytes: 2

Encoding:

0	1	0	0	0	1	0	0	immediate data

ORL direct,A

Operation: ORL

(direct) ← (direct) ∨ (A)

Bytes: 2

Encoding:

0	1	0	0	0	0	1	0	direct address

ORL direct, #data

Operation: ORL

(direct) ← (direct) ∨ #data

Bytes: 3

Encoding:

0	1	0	0	0	0	1	1
direct address							
Immediate data							

ORL C, <src-bit>

Function: Logical OR direct bit with carry flag

Description: Set the carry flag if the Boolean value is a logic 1; leave the carry in its current state otherwise. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

ORL C,bit

Operation: ORL

(C) ← (C) ∨ (bit)

Bytes: 2

Encoding:

0	1	1	1	0	0	1	0	bit address

ORL C,/bit

Operation: ORL

(C) ← (C) ∨ / (bit)

Bytes: 2

Encoding:

1	0	1	0	0	0	0	0	bit address

POP direct

Function: Pop from Stack

Description: The contents of the internal RAM location addressed by the Stack Pointer are read, and the Stack Pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected.

Operation: POP

(direct) ← ((SP))

(SP) ← (SP) - 1

Bytes: 2

Encoding:

1	1	0	1	0	0	0	0	direct address

PUSH direct

Function: Push onto Stack

Description: The Stack Pointer is incremented by one. The contents of the indicated variable are then copied into the internal RAM location addressed by the Stack Pointer. Otherwise no flags are affected.

Operation: PUSH

(SP) ← (SP) + 1

((SP)) ← (direct)

Bytes: 2

Encoding:

1	1	0	0	0	0	0	0	direct address

RET

Function: Return from subroutine

Description: RET pops the high and low-order bytes of the PC successively from the Stack, decrementing the Stack Pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.

Operation: RET

$(PC_{15\text{-}8}) \leftarrow ((SP))$

(SP) ← (SP) - 1

$(PC_{7\text{-}0}) \leftarrow ((SP))$

(SP) ← (SP) - 1

Bytes: 1

Encoding:

0	0	1	0	0	0	1	0

RETI

Function: Return from interrupt

Description: RETI pops the high and low-order bytes of the PC successively from the Stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The Stack Pointer is left decremented by two. No other registers are affected

The PSW register is *not* automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower or same-level interrupt is pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.

Operation: RETI

$(PC_{15\text{-}8}) \leftarrow ((SP))$

(SP) ← (SP) - 1

$(PC_{7\text{-}0}) \leftarrow ((SP))$

(SP) ← (SP) - 1

Bytes: 1

Encoding:

0	0	1	1	0	0	1	0

RL A

Function:	Rotate Accumulator left
Description:	The eight bits in the Accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.
Operation:	RL (An + 1) ← (An) n = 0-6 (A0) ← (A7)
Bytes:	1
Encoding:	

0	0	1	0	0	0	1	1

RLC A

Function:	Rotate Accumulator left through carry flag
Description:	The eight bits in the Accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.
Operation:	RLC (An + 1) ← (An) n = 0-6 (A0) ← (C) (C) ← (A7)
Bytes:	1
Encoding:	

0	0	1	1	0	0	1	1

RR A

Function:	Rotate Accumulator right
Description:	The eight bits in the Accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.
Operation:	RR (An) ← (An + 1) n = 0-6 (A7) ← (A0)
Bytes:	1
Encoding:	

0	0	0	0	0	0	1	1

RRC A

Function:	Rotate Accumulator right through carry flag
Description:	The eight bits in the Accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.
Operation:	RRC (An) ← (An + 1) n=0-6 (A7) ← (C) (C) ← (A0)
Bytes:	1
Encoding:	

0	0	0	1	0	0	1	1

SETB <bit>

Function: Set bit

Description: SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.

SETB bit

Operation: SETB
(bit) ← 1

Bytes: 2

Encoding:

1	1	0	1	0	0	1	0	bit address

SETB C

Operation: SETB
(C) ← 1

Bytes: 1

Encoding:

1	1	0	1	0	0	1	1

SJMP rel

Function: Short jump

Description: Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction to 127 bytes following it.

Note: Under the above conditions the instruction following SJMP will be at 102h. Therefore, the displacement byte of the instruction will be the relative offset (0123h – 0102h) = 21h . In other words, an SJMP with a displacement of 0FEh would be a one-instruction infinite loop.

Operation: SJMP
(PC) ← (PC) + 2
(PC) ← (PC) + rel

Bytes: 2

Encoding:

1	0	0	0	0	0	0	0	relative address

SUBB A, <src-byte>

Function: Subtract with borrow

Description: SUBB subtracts the indicated variable and the carry flag together from the Accumulator, leaving the result in the Accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set *before* executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the Accumulator along with the source operand).

AC (Auxiliary Carry bit) is set if a borrow is needed for bit 3, and cleared otherwise. OV (Overflow flag) is set if a borrow is needed into bit 6 but not into bit 7, or into bit 7 but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

SUBB A,Rn

Operation:	SUBB
	(A) ← (A) - (C) - (Rn)
Bytes:	1
Encoding:	

1	0	0	1	1	r	r	r

SUBB A,direct

Operation:	SUBB
	(A) ← (A) - (C) - (direct)
Bytes:	2
Encoding:	

1	0	0	1	0	1	0	1	direct address

SUBB A, @ Ri

Operation:	SUBB
	(A) ← (A) - (C) - ((Ri))
Bytes:	1
Encoding:	

1	0	0	1	0	1	1	i

SUBB A, #data

Operation:	SUBB
	(A) ← (A) - (C) - #data
Bytes:	2
Encoding:	

1	0	0	1	0	1	0	0	immediate data

SWAP A

Function:	Swap nibbles within the Accumulator
Description:	SWAP A interchanges the low and high-order nibbles (four-bit fields) of the Accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction. No flags are affected.
Operation:	SWAP
	$(A_{3-0}) \leftrightarrow (A_{7-4})$
Bytes:	1
Encoding:	

1	1	0	0	0	1	0	0

XCH A, <byte>

Function:	Exchange Accumulator with byte variable
Description:	XCH loads the Accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.

XCH A,Rn

Operation: XCH

(A) ↔ (Rn)

Bytes: 1

Encoding:

1	1	0	0	1	r	r	r

XCH A,direct

Operation: XCH

(A) ↔ (direct)

Bytes: 2

Encoding:

1	1	0	0	0	1	0	1	direct address

XCH A, @ Ri

Operation: XCH

(A) ↔ ((Ri))

Bytes: 1

Encoding:

1	1	0	0	0	1	1	i

XCHD A,@Ri

Function: Exchange digit

Description: XCHD exchanges the low-order nibble of the Accumulator (bits 3-0, generally representing a hexadecimal or BCD digit), with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.

Operation: XCHD

$(A_{3\text{-}0}) \leftrightarrow ((Ri_{3\text{-}0}))$

Bytes: 1

Encoding:

1	1	0	1	0	1	1	i

XRL <dest-byte>, <src-byte>

Function: Logical Exclusive OR for byte variables

Description: XRL performs the bit wise logical Exclusive OR operation between the indicated variables, storing the results in the destination. No flags are affected (except P (Parity bit), if <dest-byte> = A).

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be either the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

XRL A,Rn

Operation: XRL2

(A) ← (A) ∀ (Rn)

Bytes: 1

Encoding:

0	1	1	0	1	r	r	r

XRL A,direct

Operation: XRL

(A) ← (A) ∀ (direct)

Bytes: 2

Encoding:

0	1	1	0	0	1	0	1	direct address

XRL A, @ Ri

Operation: XRL

(A) ← (A) ∀ ((Ri))

Bytes: 1

Encoding:

0	1	1	0	0	1	1	i

XRL A, #data

Operation: XRL

(A) ← (A) ∀ #data

Bytes: 2

Encoding:

0	1	1	0	0	1	0	0	immediate data

XRL direct,A

Operation: XRL

(direct) ← (direct) ∀ (A)

Bytes: 2

Encoding:

0	1	1	0	0	0	1	0	direct address

XRL direct, #data

Operation: XRL

(direct) ← (direct) ∀ #data

Bytes: 3

Encoding:

0	1	1	0	0	0	1	1
direct address							
immediate data							

Instruction Timing

With the exception of MUL and DIV, all instructions in the set take one or two instruction cycles to complete. A TSK51x instruction cycle consists of 12 clock cycles. Each clock cycle forms a CPU cycle. Therefore, an instruction cycle consists of six CPU states and two phases. Various events occur in each CPU cycle, depending on the type of instruction being executed.

Program Memory Timing

The execution of instruction N is performed during the fetch of instruction N+1.

Internal Program Memory Read Cycle

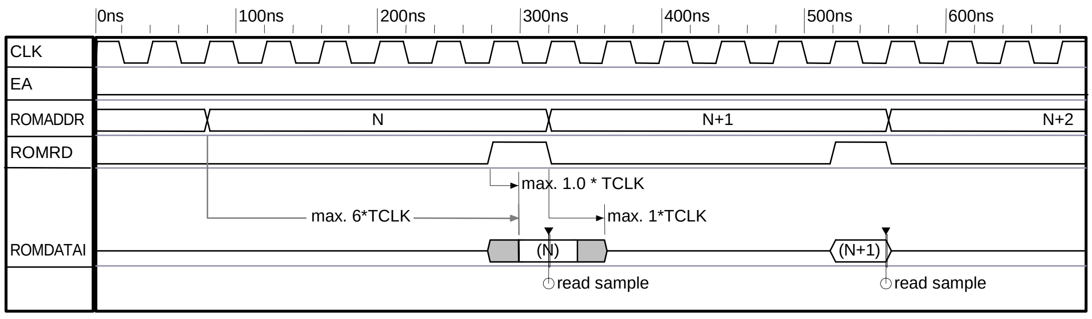


Figure 14. Internal Program memory Read cycle

Note:	TCLK	- time period of CLK signal
	N	- address of current instruction to be executed
	(N)	- instruction fetched from address N
	N+1	- address of next instruction to be executed
	read sample	- point at which data is read from bus into the internal register.

External Program Memory Read Cycle

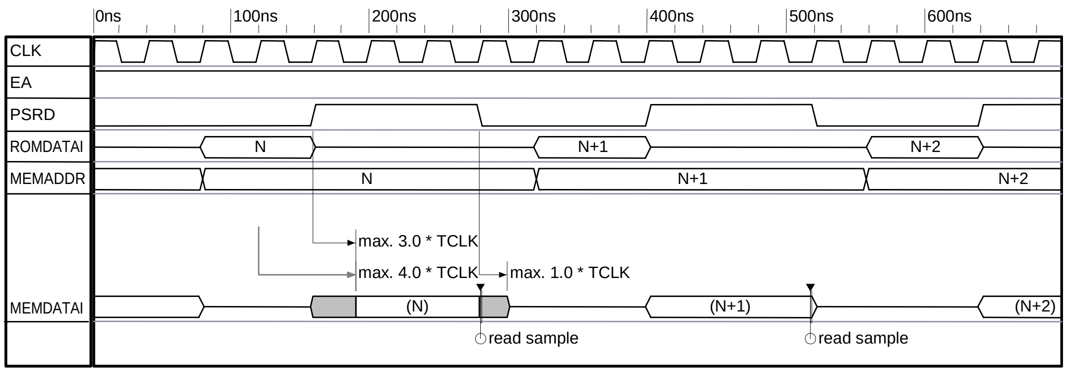


Figure 15. External Program memory Read cycle

Note:	addrbus	- externally latched address bus
	TCLK	- time period of CLK signal
	N	- address of current instruction to be executed
	N+1	- address of next instruction to be executed
	(N)	- instruction fetched from address N
	read sample	- point at which data is read from bus into the internal register.

Data Memory Timing

Internal Data Memory Read Cycle

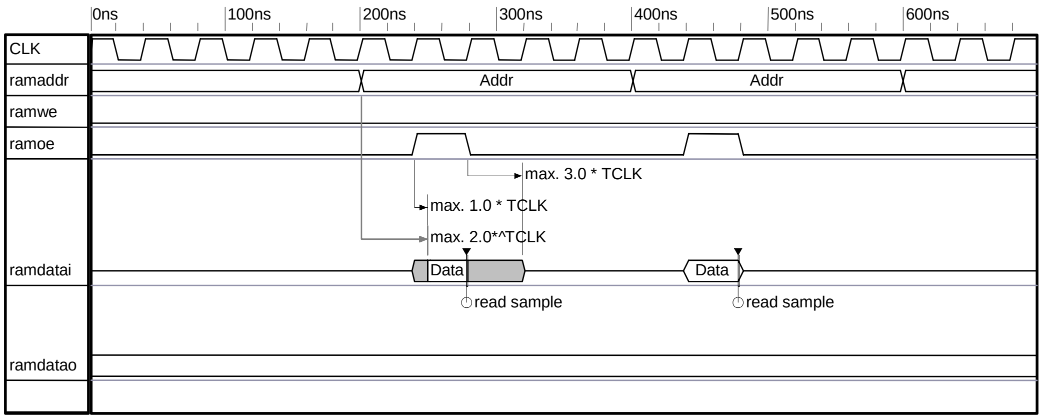


Figure 16. Internal Data memory Read cycle

Note:		
	TCLK	- time period of CLK signal
	Addr	- address of memory cell
	Data	- data to be read from address Addr
	read sample	- point at which data is read from bus into the internal register.

Internal Data Memory Write Cycle

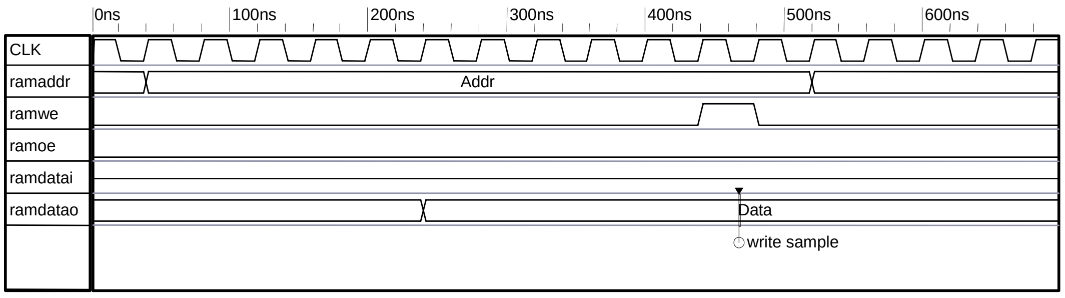


Figure 17. Internal Data memory Write cycle

Note:		
	TCLK	- time period of CLK signal
	Addr	- address of memory cell
	Data	- data to be written into address Addr
	write sample	- point at which data is written from the bus into memory.

External Data Memory Read Cycle

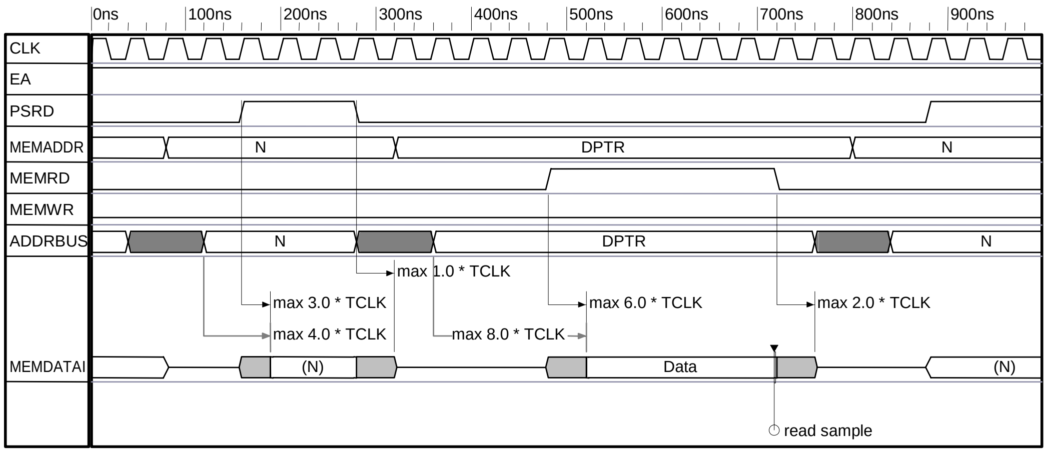


Figure 18. External Data memory Read cycle

Note:	addrbus	- externally latched address bus
	TCLK	- time period of CLK signal
	N	- address of current instruction to be executed
	N+1	- address of next instruction to be executed
	(N)	- instruction fetched from address N
	DPTR	- address of data loaded into Data Pointer
	Data	- data to be read from address referenced by DPTR
	read sample	- point at which data is read from bus into the internal register.

External Data Memory Write Cycle

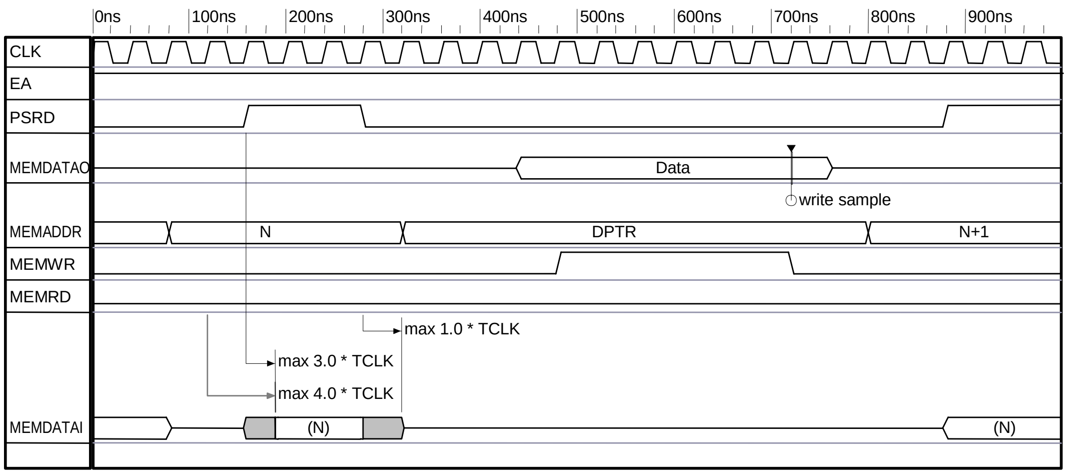


Figure 19. External Data memory Write cycle

Note:	addrbus	- externally latched address bus
	TCLK	- time period of CLK signal
	N	- address of current instruction to be executed
	N+1	- address of next instruction to be executed
	(N)	- instruction fetched from address N
	DPTR	- address of data loaded into Data Pointer
	Data	- data to be written into address referenced by DPTR
	write sample	- point at which data is written from the bus into memory.

External Special Function Registers Timing

External Special Function Register Read Cycle

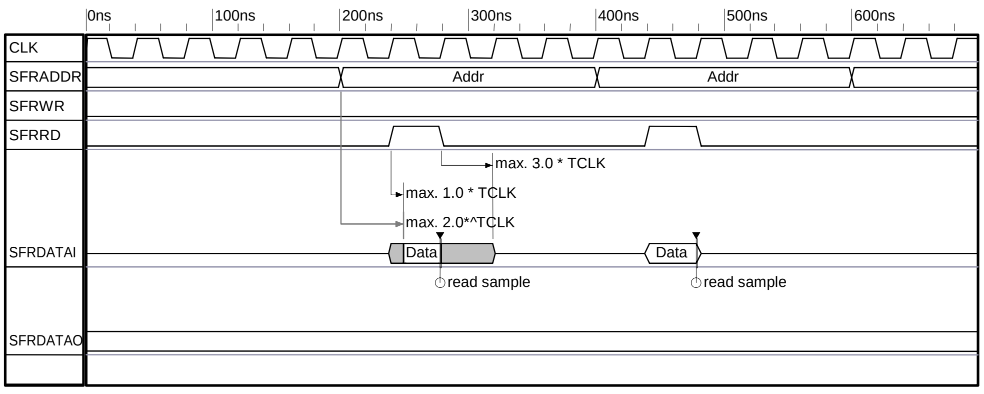


Figure 20. External special function register Read cycle

Note:		
	TCLK	- time period of CLK signal
	Addr	- address of special function register
	Data	- data to be read from address Addr
	read sample	- point at which data is read from bus into the internal register

External Special Function Register Write Cycle

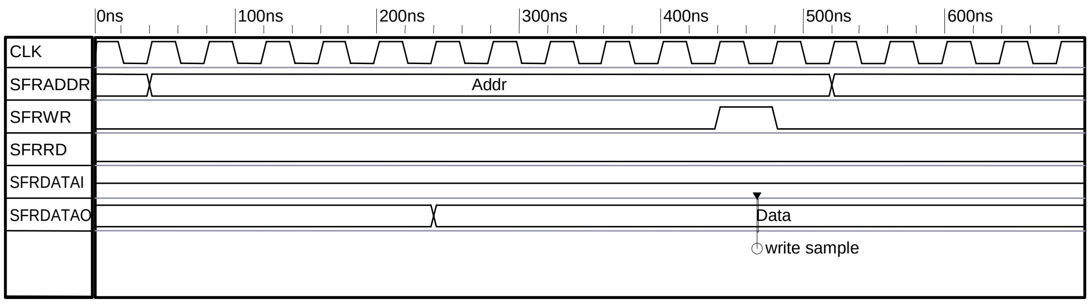


Figure 21. External special function registers Write cycle

Note:		
	TCLK	- time period of CLK signal
	Addr	- address of special function register
	Data	- data to be written into address Addr
	write sample	- point at which data is written from the bus into register.

Revision History

Date	**Version No.**	**Revision**
22-Jan-2004	1.0	New product release
22-Oct-2004	1.1	Modifications to Timer/Counter information, polarity updates for Interrupt and Timer signals in main Pinout table. Changes to On-Chip Debugging, including addition of the Nexus Debugger panel.
08-Feb-2005	1.2	Modifications to debug panel information in On-Chip Debugging section.
09-May-2005	1.3	Updated for SP4
12-Dec-2005	1.4	Path references updated for Altium Designer 6
13-Mar-2008	2.0	Updated for Altium Designer Summer 08