# Core8051

DirectCore

## Product Summary

### Intended Use

- Embedded System Control
- Communication System Control
- I/O Control

### Key Features

- 100% ASM51 Compatible Instruction Set[1]
- Control Unit
  - Eight-bit Instruction Decoder
  - Reduced Instruction Time of up to 12 Cycles
- Arithmetic Logic Unit
  - Eight-bit Arithmetic and Logical Operations
  - Boolean Manipulations
  - Eight by Eight-bit Multiplication and Eight by Eight-bit Division
- 32-bit I/O Ports
  - Four Eight-bit I/O Ports
  - Alternate Port Functions, such as External Interrupts, Provide Extra Port Pins when Compared with the Standard 8051
- Serial Port Interfaces
  - Simultaneous Transmit and Receive
  - Synchronous Mode, Fixed Baud Rate
  - Eight-bit UART Mode, Variable Baud Rate
  - Nine-bit UART Mode, Fixed Baud Rate
  - Nine-bit UART Mode, Variable Baud Rate
  - Multiprocessor Communication
- Two 16-bit Timer/Counters
- Interrupt Controller
  - Four Priority Levels with 13 Interrupt Sources
- Internal Data Memory Interface
  - Can Address up to 256B of Data Memory Space
- External Memory Interface
  - Can Address up to 64kB of External Program Memory
  - Can Address up to 64kB of External Data Memory
  - Demultiplexed Address/Data Bus Enables Easy Connection to Memories
  - Variable Length MOVX to Access Fast/Slow RAM or Peripherals
  - Wait Cycles to Access Fast/Slow ROM
  - Dual Data Pointer to Fast Data Block Transfer
- Special Function Register (SFR) Interface
  - Services up to 101 External SFRs
- Optional On-Chip Instrumentation (OCI) Debug Logic
- Supports all Major Actel Device Families

### Targeted Devices

- ProASIC$^{PLUS}$ Family
- Axcelerator Family
- SX-A Family
- RT54SX-S Family
- RTAX-S Family

### Core Deliverables

- Evaluation Version
  - Compiled RTL Simulation Model Fully Supported in Actel's Libero™ Integrated Design Environment (IDE)
- Netlist Version
  - Structural Verilog and VHDL Netlists (with and without I/O Pads) Compatible with Actel's Designer Software Place-and-Route Tool
  - Compiled RTL Simulation Model Fully Supported in Actel's Libero IDE
- RTL Version
  - Verilog and VHDL Core Source Code
  - Core Synthesis Scripts
- Testbench (Verilog and VHDL)

### Synthesis and Simulation Support

- Synthesis
  - Synplicity
  - Synopsys (Design Compiler, FPGA Compiler, FPGA Express)
  - Exemplar
- Simulation
  - OVI - Compliant Verilog Simulators
  - Vital - Compliant VHDL Simulators

### Core Verification

- Comprehensive VHDL and Verilog Testbenches
- Users Can Easily Add Custom Tests by Modifying the User Testbench Using the Existing Format

*1. For more information, see the Core8051 Instruction Set Details Advanced User's Guide*

## Table of Contents



### General Description

The Core8051 macro is a high performance single-chip eight-bit microcontroller. It is a fully functional eight-bit embedded controller that executes all ASM51 instructions and has the same instruction set as the 80C31. Core8051 provides software and hardware interrupts, a serial port, and two timers.

The Core8051 architecture eliminates redundant bus states and implements parallel execution of fetch and execution phases. Since a cycle is aligned with memory fetch when possible, most of the one byte instructions are performed in a single cycle. Core8051 uses one clock per cycle. This leads to an average performance improvement rate of 8.0 (in terms of MIPS) with respect to the Intel device working with the same clock frequency.

The original 8051 had a 12-clock architecture. A machine cycle needed 12 clocks and most instructions were either one or two machine cycles. Therefore, the 8051 used either 12 or 24 clocks for each instruction, except for the MUL and DIV instructions. Furthermore, each cycle in the 8051 used two memory fetches. In many cases, the second fetch was a “dummy” fetch and extra clocks were wasted.

Table 1 shows the speed advantage of Core8051 over the standard 8051. A speed advantage of 12 in the first column means that Core8051 performs the same instruction 12 times faster than the standard 8051. The second column in Table 1 lists the number of types of instructions that have the given speed advantage. The third column lists the total number of instructions that have the given speed advantage. The third column can be thought of as a subcategory of the second column. For example, in the last row, there are two types of instructions that have a three time speed advantage over the classic 8051, for which there are nine explicit instructions.

***Table 1 •*** *Core8051 Speed Advantage Summary*

| Speed advantage | Number of instruction types | Number of instructions (opcodes) |
|---|---|---|
| 24 | 1 | 1 |
| 12 | 27 | 83 |
| 9.6 | 2 | 2 |
| 8 | 16 | 38 |
| 6 | 44 | 89 |
| 4.8 | 1 | 2 |
| 4 | 18 | 31 |
| 3 | 2 | 9 |
| Average: 8.0 | Sum: 111 | Sum: 255 |

The average speed advantage is 8.0. However, the real speed improvement seen in any system will depend on the instruction mix.

Core8051 consists of the following primary blocks:

1. Memory Control Block – Logic that Controls Program and Data Memory
2. Control Processor Block – Main Controller Logic
3. RAM and SFR Control Block
4. ALU – Arithmetic Logic Unit
5. Reset Control Block – Provides Reset Condition Circuitry
6. Clock Control Block
7. Timer 0 and 1 Block
8. ISR – Interrupt Service Routine Block
9. Serial Port Block
10. Port Registers Block
11. PMU – Power Management Unit Block
12. OCI block – On-Chip Instrumentation Logic for Debug Capabilities

Figure 1 on page 3 shows the primary blocks of Core8051.

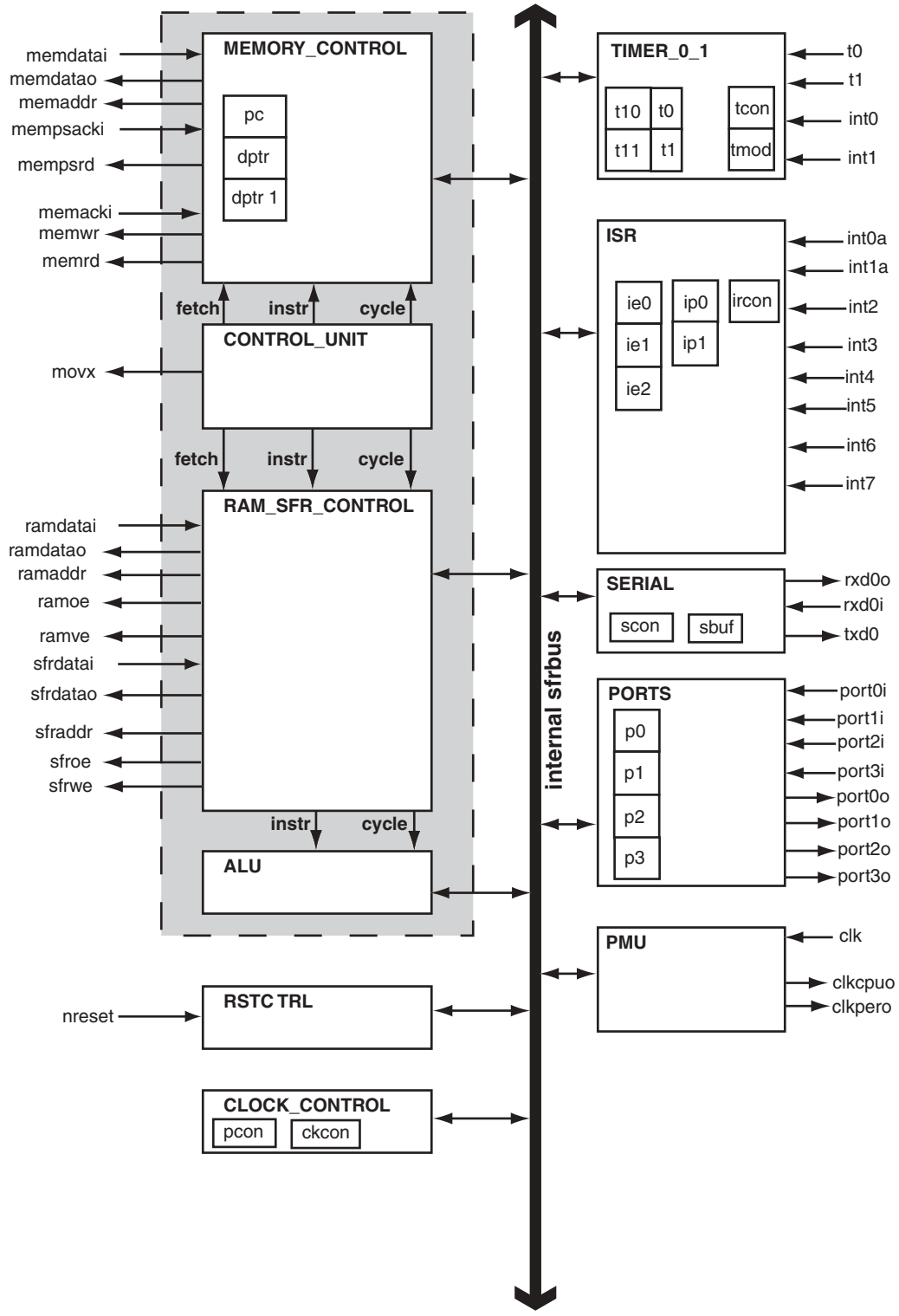


***Figure 1*** • *Core8051 Block Diagram*

## Core8051 Device Requirements

Core8051 has been implemented in several of Actel's device families. A summary of the implementation data is listed in Tables 2 to 4. Table 2 lists implementation data without OCI logic.

***Table 2 •*** *Core8051 Device Utilization and Performance - No OCI*

| Family | Cells or Tiles | | Utilization | | Performance |
|---|---|---|---|---|---|
| | Sequential | Combinatorial | Device | Total | |
| ProASIC$^{PLUS}$ | 609 | 5044 | APA150-STD | 92% | 20MHz |
| Axcelerator | 653 | 2848 | AX500-3 | 44% | 45MHz |
| SX-A | 656 | 3014 | A54SX72A-3 | 61% | 30MHz |
| RT54SX-S | 656 | 3014 | RT54SX72S-1 | 61% | 12MHz |

***Note:*** *Data in this table were achieved using typical synthesis and layout settings.*

Table 3 lists implementation data with OCI logic (no trace memory and no hardware triggers).

***Table 3 •*** *Core8051 Device Utilization and Performance - OCI without Trace Memory and Hardware Trigger*

| Family | Cells or Tiles | | Utilization | | Performance |
|---|---|---|---|---|---|
| | Sequential | Combinatorial | Device | Total | |
| ProASIC$^{PLUS}$ | 723 | 5289 | APA300-STD | 74% | 19MHz |
| Axcelerator | 766 | 2939 | AX500-3 | 46% | 45MHz |
| SX-A | 758 | 3062 | A54SX72A-3 | 64% | 25MHz |
| RT54SX-S | 760 | 3051 | RT54SX72S-1 | 64% | 11MHz |

***Note:*** *Data in this table were achieved using typical synthesis and layout settings.*

Table 4 lists implementation data with OCI logic (256-word trace memory and one hardware trigger).

***Table 4 •*** *Core8051 Device Utilization and Performance - OCI with 256-Word Trace Memory and One Hardware Trigger*

| Family | Cells or Tiles | | Utilization | | Performance |
|---|---|---|---|---|---|
| | Sequential | Combinatorial | Device | Total | |
| ProASIC$^{PLUS}$ | 827 | 5905 | APA300-STD | 83% | 15MHz |
| Axcelerator | 885 | 3292 | AX500-3 | 52% | 37MHz |

***Note:*** *Data in this table were achieved using typical synthesis and layout settings.*

## Core8051 Verification

The comprehensive verification simulation testbench (included with the Netlist and RTL versions of the core) verifies correct operation of the Core8051 macro. The verification testbench applies several tests to the Core8051 macro, including:

- Operation Code Tests
- Peripheral Tests
- Miscellaneous Tests

Using the supplied user testbench as a guide, the user can easily customize the verification of the core by adding or removing tests.

## I/O Signal Descriptions

The port signals for the Core8051 macro are defined in Table 5 and illustrated in Figure 2. All signals are either “Input” (input-only) or “Output” (output-only).

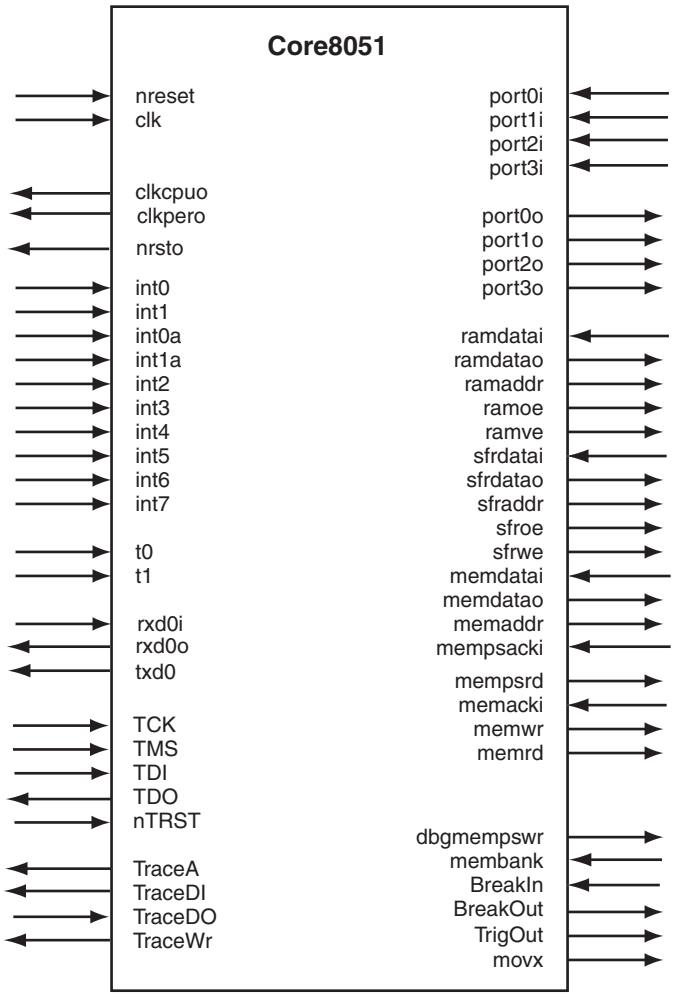


***Figure 2*** • *Core8051 I/O Signal Diagram*

***Table 5*** • *Core8051 Pin Description*

| Name | Type | Polarity/Bus size | Description |
|---|---|---|---|
| port0i | Input | 8 | **Port 0** |
| port0o | Output | 8 | Eight-bit bidirectional I/O port with separated inputs and outputs |
| port1i | Input | 8 | **Port 1** |
| port1o | Output | 8 | Eight-bit bidirectional I/O port with separated inputs and outputs |
| port2i | Input | 8 | **Port 2** |
| port2o | Output | 8 | Eight-bit bidirectional I/O port with separated inputs and outputs |
| port3i | Input | 8 | **Port 3** |
| port3o | Output | 8 | Eight-bit bidirectional I/O port with separated inputs and outputs |

***Table 5 •*** *Core8051 Pin Description (Continued)*

| Name | Type | Polarity/Bus size | Description |
|---|---|---|---|
| clk | Input | Rise | **Clock** pulse for internal clock counters and all synchronous circuits |
| clkcpuo | Output | Rise | **Engine Clock Output**<br>This is a gated globally buffered version of the clk input that remains at logic 0 when Core8051 enters the IDLE or STOP mode. This signal can be connected to logic outside Core8051 |
| clkpero | Output | Rise | **Peripheral Clock Output**<br>This is a gated globally buffered version of the clk input that remains at logic 0 when Core8051 enters the STOP mode. This signal can be connected to logic outside Core8051 |
| nreset | Input | Low | **Hardware Reset Input**<br>A logic 0 on this pin for two clock cycles while the oscillator is running resets the device |
| nrsto | Output | Low | **Peripheral Reset Output**<br>This globally buffered signal can be connected to logic outside Core8051 to provide an active-low asynchronous reset to peripherals |
| movx | Output | High | **Movx** instruction executing |
| | | | **On-Chip Debug Interface (Optional)** |
| TCK | Input | Rise | JTAG test clock. If OCI is not used, connect to logic 1 |
| TMS | Input | High | JTAG test mode select. If OCI is not used, connect to logic 0 |
| TDI | Input | High | JTAG test data in. If OCI is not used, connect to logic 0 |
| TDO | Output | High | JTAG test data out |
| nTRST | Input | Low | JTAG test reset. If OCI is not used, connect to logic 1 |
| dbgmempswr | Output | High | Optional debug program storage write |
| membank | Input | 4 | Optional code memory bank selection. If not used, connect to logic 0 values |
| BreakIn | Input | High | Break bus input. When sampled high, a breakpoint is generated. If not used, connect to logic 0 |
| BreakOut | Output | High | Break bus output. This will be driven high when Core8051 stops emulation. This can be connected to an open-drain Break bus that connects to multiple processors, so that when any CPU stops, all others on the bus are stopped within a few clock cycles |
| TrigOut | Output | High | Trigger output. This signal can be optionally connected to external test equipment to cross-trigger with internal Core8051 activity |
| TraceA | Output | 8 | Trace address outputs. This bus should be connected to external RAM address pins for trace debug memory |
| TraceDI | Output | 20 | Trace data to external synchronous RAM data input pins for trace debug memory |
| TraceDO | Input | 20 | Trace data from external synchronous RAM data output pins for trace debug memory. If OCI is not used, connect to logic 0 values |
| TraceWr | Output | High | Trace write signal to external synchronous RAM write enable for trace debug memory |

***Table 5 •*** *Core8051 Pin Description (Continued)*

| Name | Type | Polarity/Bus size | Description |
|---|---|---|---|
| | | | **External Interrupt Inputs** |
| int0 | Input | Low/Fall | External interrupt 0 |
| int1 | Input | Low/Fall | External interrupt 1 |
| int0a | Input | High | External interrupt 0a |
| int1a | Input | High | External interrupt 1a |
| int2 | Input | High | External interrupt 2 |
| int3 | Input | High | External interrupt 3 |
| int4 | Input | High | External interrupt 4 |
| int5 | Input | High | External interrupt 5 |
| int6 | Input | High | External interrupt 6 |
| int7 | Input | High | External interrupt 7 |
| | | | **Serial Port Interface** |
| rxdi | Input | – | Serial port receive data |
| rxdo | Output | – | Serial port transmit data in mode 0 |
| txd | Output | – | Serial port transmit data or data clock in mode 0 |
| | | | **Timer Inputs** |
| t0 | Input | Fall | Timer 0 external input |
| t1 | Input | Fall | Timer 1 external input |
| | | | **External Memory Interface** |
| mempsacki | Input | High | Program memory read acknowledge |
| memacki | Input | High | Data memory acknowledge |
| memdatai | Input | 8 | Memory data input |
| memdatao | Output | 8 | Memory data output |
| memaddr | Output | 16 | Memory address |
| mempsrd | Output | High | Program store read enable |
| memwr | Output | High | Data memory write enable |
| memrd | Output | High | Data memory read enable |
| | | | **Internal Data Memory Interface** |
| ramdatai | Input | 8 | Data bus input |
| ramdatao | Output | 8 | Data bus output |
| ramaddr | Output | 8 | Data file address |
| ramwe | Output | High | Data file write enable |
| ramoe | Output | High | Data file output enable |
| | | | **External Special Function Registers Interface** |
| sfrdatai | Input | 8 | SFR data bus input |
| sfrdatao | Output | 8 | SFR data bus output |
| sfraddr | Output | 7 | SFR address |
| sfrwe | Output | High | SFR write enable |
| sfroe | Output | High | SFR output enable |

## Memory Organization

The Core8051 microcontroller utilizes the Harvard architecture, with separate code and data spaces.

Memory organization in Core8051 is similar to that of the industry standard 8051. There are three memory areas, as shown in Figure 3:

- Program Memory (Internal RAM, External RAM, or External ROM)
- External Data Memory (External RAM)
- Internal Data Memory (Internal RAM)

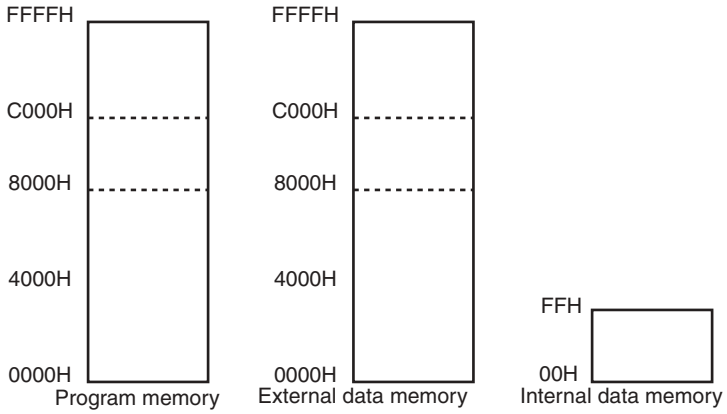


***Figure 3 •*** *Core8051 Memory Map*

### Program Memory

Core8051 can address up to 64kB of program memory space, from 0000H to FFFFH. The External Bus Interface services program memory when the mempsrd signal is active. Program memory is read when the CPU performs fetching instructions or MOVC.

After reset, the CPU starts program execution from location 0000H. The lower part of the program memory includes interrupt and reset vectors. The interrupt vectors are spaced at eight-byte intervals, starting from 0003H.

Program memory can be implemented as Internal RAM, External RAM, External ROM, or a combination of all three.

### External Data Memory

Core8051 can address up to 64kB of external data memory space, from 0000H to FFFFH. The External Bus Interface services program memory when the memrd signal is active. Writing to external program memory is only supported in debug mode using the OCI logic block and external debugger hardware and software. Core8051 writes into external data memory when the CPU executes a MOVX @Ri,A or MOVX @DPTR,A instructions. The external data memory is read when the CPU executes MOVX A,@Ri or MOVX A,@DPTR instructions.

There is improved variable length of the MOVX instructions to access fast or slow external RAM and external peripherals. The three low-ordered bits of the ckcon register control stretch memory cycles. Setting ckcon stretch bits to logic 1 values, enables access to very slow external RAM or external peripherals.

Table 6 on page 9 shows how the External Memory Interface signals change when stretch values are set from zero to seven. The widths of the signals are counted in clk cycles. The reset state of the ckcon register has a stretch value equal to one (001), which enables MOVX instructions to be performed with a single stretch clock cycle inserted.

***Table 6*** • *Stretch Memory Cycle Width*

| ckcon register | | | Stretch value | Read signal width | | Write signal width | |
|---|---|---|---|---|---|---|---|
| ckcon.2 | ckcon.1 | ckcon.0 | | memaddr | memrd | memaddr | memwr |
| 0 | 0 | 0 | 0 | 1 | 1 | 2 | 1 |
| 0 | 0 | 1 | 1 | 2 | 2 | 3 | 1 |
| 0 | 1 | 0 | 2 | 3 | 3 | 4 | 2 |
| 0 | 1 | 1 | 3 | 4 | 4 | 5 | 3 |
| 1 | 0 | 0 | 4 | 5 | 5 | 6 | 4 |
| 1 | 0 | 1 | 5 | 6 | 6 | 7 | 5 |
| 1 | 1 | 0 | 6 | 7 | 7 | 8 | 6 |
| 1 | 1 | 1 | 7 | 8 | 8 | 9 | 7 |

There are two types of instructions, one provides an eight-bit address to the external data RAM, the other a 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 eight-bit address. The eight high ordered bits of address are stuck at zero. Eight bits are sufficient for external l/O expansion decoding or a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins are controlled by an output instruction preceding the MOVX.

In the second type of MOVX instructions, the data pointer generates a 16-bit address. This form is faster and more efficient when accessing very large data arrays (up to 64kB), since no additional instructions are needed to set up the output ports.

In some situations, it is possible to mix the two MOVX types. A large RAM array with its high-order address lines can be addressed via the data pointer, or with code to output high-order address bits, to any port followed by a MOVX instruction using R0 or R1.

### Internal Data Memory

The internal data memory interface services up to 256 bytes of off-core data memory. The internal data memory address is always one byte wide. The memory space is 256 bytes large (00H to FFH) and can be accessed by direct or indirect addressing. The SFRs occupy the upper 128 bytes. This SFR area is available only by direct addressing. Indirect addressing accesses the upper 128 bytes of internal RAM.

The lower 128 bytes contain work registers and bit-addressable memory. The lower 32 bytes form four banks of eight registers (R0-R7). Two bits on the program memory status word (PSW) select which bank is in use. The next 16 bytes form a block of bit-addressable memory space at bit addressees 00H-7FH. All of the bytes in the lower 128 bytes are accessible through direct or indirect addressing.

## Special Function Registers

### Internal Special Function Registers

A map of the internal Special Function Registers is shown in Table 7. Only a few addresses are occupied, the others are not implemented. Read access to unimplemented addresses will return undefined data, while write access will have no effect.

***Table 7 •*** *Internal Special Function Register Memory Map*

| Hex | Bin | | | | | | | | Hex |
|---|---|---|---|---|---|---|---|---|---|
| | X000 | X001 | X010 | X011 | X100 | X101 | X110 | X111 | |
| F8 | – | – | – | – | – | – | – | – | FF |
| F0 | b | – | – | – | – | – | – | – | F7 |
| E8 | – | – | – | – | – | – | – | – | EF |
| E0 | acc | – | – | – | – | – | – | – | E7 |
| D8 | – | – | – | – | – | – | – | – | DF |
| D0 | psw | – | – | – | – | – | – | – | D7 |
| C8 | – | – | – | – | – | – | – | – | CF |
| C0 | – | – | – | – | – | – | – | – | C7 |
| B8 | ien1 | ip1 | – | – | – | – | – | – | BF |
| B0 | p3 | – | – | – | – | – | – | – | B7 |
| A8 | ien0 | ip0 | – | – | – | – | – | – | AF |
| A0 | p2 | – | – | – | – | – | – | – | A7 |
| 98 | scon | sbuf | – | – | – | – | – | – | 9F |
| 90 | p1 | – | dps | – | – | – | – | – | 97 |
| 88 | tcon | tmod | tl0 | tl1 | th0 | th1 | ckcon | – | 8F |
| 80 | p0 | sp | dpl | dph | dpl1 | dph1 | – | pcon | 87 |

The reset value for of each of the predefined special function registers is listed in Table 8.

***Table 8** • Special Function Register Reset Values*

| Register | Location | Reset value | Description |
|---|---|---|---|
| p0 | 80h | FFh | Port 0 |
| sp | 81h | 07h | Stack Pointer |
| dpl | 82h | 00h | Data Pointer Low 0 |
| dph | 83h | 00h | Data Pointer High 0 |
| dpl1 | 84h | 00h | Dual Data Pointer Low 1 |
| dph1 | 85h | 00h | Dual Data Pointer High 1 |
| pcon | 87h | 00h | Power Control |
| tcon | 88h | 00h | Timer/Counter Control |
| tmod | 89h | 00h | Timer Mode Control |
| tl0 | 8Ah | 00h | Timer 0, low byte |
| tl1 | 8Bh | 00h | Timer 1, high byte |
| th0 | 8Ch | 00h | Timer 0, low byte |
| th1 | 8Dh | 00h | Timer 1, high byte |
| ckcon | 8Eh | 01h | Clock Control (Stretch=1) |
| p1 | 90h | FFh | Port 1 |
| dps | 92h | 00h | Data Pointer Select Register |
| scon | 98h | 00h | Serial Port 0, Control Register |
| sbuf | 99h | 00h | Serial Port 0, Data Buffer |
| p2 | A0h | 00h | Port 2 |
| ien0 | A8h | 00h | Interrupt Enable Register 0 |
| ien1 | B8h | 00h | Interrupt Enable Register 1 |
| p3 | B0h | FFh | Port 3 |
| ip0 | A9h | 00h | Interrupt Enable Register 0 |
| ip1 | B9h | 00h | Interrupt Enable Register 1 |
| psw | D0h | 00h | Program Status Word |

### External Special Function Registers

The external SFR interface services up to 101 off-core special function registers. The off-core peripherals can use all addresses from the SFR address space range 80H to FFH except for those that are already implemented inside the core.

When a read instruction occurs with a SFR address that has been implemented both inside and outside the core, the read will return the contents of the internal SFR.

When a write instruction occurs with a SFR that has been implemented both inside and outside the core, the value of the external SFR is overwritten.

## Instruction Set

All Core8051 instructions are binary code compatible and perform the same functions as they do with the industry standard 8051. Tables 9 and 10 contain notes for mnemonics used in the various Instruction Set tables. In Tables 11 through 15, the instructions are ordered in functional groups. In Table 16, the instructions are ordered in the hexadecimal order of their operation code. For more detailed information about the Core8051 instruction set, refer to the *Core8051 Instruction Set Advanced User's Guide*.

***Table 9 •*** *Notes on Data Addressing Modules*

| | |
|---|---|
| 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 | Eight-bit constant included in instruction |
| #data 16 | 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 10 •*** *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 eight-bit offset byte. Range is from plus 127 to minus 128 bytes, relative to the first byte of the following instruction |

## Functional Ordered Instructions

Tables 11 through 15 list the Core8051 instructions, grouped according to function.

***Table 11*** • *Arithmetic Operations*

| Mnemonic | Description | Byte | Cycle |
|---|---|---|---|
| ADD A,Rn | Adds the register to the accumulator | 1 | 1 |
| ADD A,direct | Adds the direct byte to the accumulator | 2 | 2 |
| ADD A,@Ri | Adds the indirect RAM to the accumulator | 1 | 2 |
| ADD A,#data | Adds the immediate data to the accumulator | 2 | 2 |
| ADDC A,Rn | Adds the register to the accumulator with a carry flag | 1 | 1 |
| ADDC A,direct | Adds the direct byte to A with a carry flag | 2 | 2 |
| ADDC A,@Ri | Adds the indirect RAM to A with a carry flag | 1 | 2 |
| ADDC A,#data | Adds the immediate data to A with carry a flag | 2 | 2 |
| SUBB A,Rn | Subtracts the register from A with a borrow | 1 | 1 |
| SUBB A,direct | Subtracts the direct byte from A with a borrow | 2 | 2 |
| SUBB A,@Ri | Subtracts the indirect RAM from A with a borrow | 1 | 2 |
| SUBB A,#data | Subtracts the immediate data from A with a borrow | 2 | 2 |
| INC A | Increments the accumulator | 1 | 1 |
| INC Rn | Increments the register | 1 | 2 |
| INC direct | Increments the direct byte | 2 | 3 |
| INC @Ri | Increments the indirect RAM | 1 | 3 |
| DEC A | Decrements the accumulator | 1 | 1 |
| DEC Rn | Decrements the register | 1 | 1 |
| DEC direct | Decrements the direct byte | 1 | 2 |
| DEC @Ri | Decrements the indirect RAM | 2 | 3 |
| INC DPTR | Increments the data pointer | 1 | 3 |
| MUL A,B | Multiplies A and B | 1 | 5 |
| DIV A,B | Divides A by B | 1 | 5 |
| DA A | Decimal adjust accumulator | 1 | 1 |

***Table 12 •*** *Logic Operations*

| Mnemonic | Description | Byte | Cycle |
|---|---|---|---|
| ANL A,Rn | AND register to accumulator | 1 | 1 |
| ANL A,direct | AND direct byte to accumulator | 2 | 2 |
| ANL A,@Ri | AND indirect RAM to accumulator | 1 | 2 |
| ANL A,#data | AND immediate data to accumulator | 2 | 2 |
| ANL direct,A | AND accumulator to direct byte | 2 | 3 |
| ANL direct,#data | AND immediate data to direct byte | 3 | 4 |
| ORL A,Rn | OR register to accumulator | 1 | 1 |
| ORL A,direct | OR direct byte to accumulator | 2 | 2 |
| ORL A,@Ri | OR indirect RAM to accumulator | 1 | 2 |
| ORL A,#data | OR immediate data to accumulator | 2 | 2 |
| ORL direct,A | OR accumulator to direct byte | 2 | 3 |
| ORL direct,#data | OR immediate data to direct byte | 3 | 4 |
| XRL A,Rn | Exclusive OR register to accumulator | 1 | 1 |
| XRL A,direct | Exclusive OR direct byte to accumulator | 2 | 2 |
| XRL A,@Ri | Exclusive OR indirect RAM to accumulator | 1 | 2 |
| XRL A,#data | Exclusive OR immediate data to accumulator | 2 | 2 |
| XRL direct,A | Exclusive OR accumulator to direct byte | 2 | 3 |
| XRL direct,#data | Exclusive OR immediate data to direct byte | 3 | 4 |
| CLR A | Clears the accumulator | 1 | 1 |
| CPL A | Complements the accumulator | 1 | 1 |
| RL A | Rotates the accumulator left | 1 | 1 |
| RLC A | Rotates the accumulator left through carry | 1 | 1 |
| RR A | Rotates the accumulator right | 1 | 1 |
| RRC A | Rotates the accumulator right through carry | 1 | 1 |
| SWAP A | Swaps nibbles within the accumulator | 1 | 1 |

***Table 13** • Data Transfer Operations*

| Mnemonic | Description | Byte | Cycle |
|---|---|---|---|
| MOV A,Rn | Moves the register to the accumulator | 1 | 1 |
| MOV A,direct | Moves the direct byte to the accumulator | 2 | 2 |
| MOV A,@Ri | Moves the indirect RAM to the accumulator | 1 | 2 |
| MOV A,#data | Moves the immediate data to the accumulator | 2 | 2 |
| MOV Rn,A | Moves the accumulator to the register | 1 | 2 |
| MOV Rn,direct | Moves the direct byte to the register | 2 | 4 |
| MOV Rn,#data | Moves the immediate data to the register | 2 | 2 |
| MOV direct,A | Moves the accumulator to the direct byte | 2 | 3 |
| MOV direct,Rn | Moves the register to the direct byte | 2 | 3 |
| MOV direct,direct | Moves the direct byte to the direct byte | 3 | 4 |
| MOV direct,@Ri | Moves the indirect RAM to the direct byte | 2 | 4 |
| MOV direct,#data | Moves the immediate data to the direct byte | 3 | 3 |
| MOV @Ri,A | Moves the accumulator to the indirect RAM | 1 | 3 |
| MOV @Ri,direct | Moves the direct byte to the indirect RAM | 2 | 5 |
| MOV @Ri,#data | Moves the immediate data to the indirect RAM | 2 | 3 |
| MOV DPTR,#data16 | Loads the data pointer with a 16-bit constant | 3 | 3 |
| MOVC A,@A + DPTR | Moves the code byte relative to the DPTR to the accumulator | 1 | 3 |
| MOVC A,@A + PC | Moves the code byte relative to the PC to the accumulator | 1 | 3 |
| MOVX A,@Ri | Moves the external RAM (eight-bit address) to A | 1 | 3-10 |
| MOVX A,@DPTR | Moves the external RAM (16-bit address) to A | 1 | 3-10 |
| MOVX @Ri,A | Moves A to the external RAM (eight-bit address) | 1 | 4-11 |
| MOVX @DPTR,A | Moves A to the external RAM (16-bit address) | 1 | 4-11 |
| PUSH direct | Pushes the direct byte onto the stack | 2 | 4 |
| POP direct | Pops the direct byte from the stack | 2 | 3 |
| XCH A,Rn | Exchanges the register with the accumulator | 1 | 2 |
| XCH A,direct | Exchanges the direct byte with the accumulator | 2 | 3 |
| XCH A,@Ri | Exchanges the indirect RAM with the accumulator | 1 | 3 |
| XCHD A,@Ri | Exchanges the low-order nibble indirect RAM with A | 1 | 3 |

***Table 14 •*** *Boolean Manipulation Operations*

| Mnemonic | Description | Byte | Cycle |
|---|---|---|---|
| CLR C | Clears the carry flag | 1 | 1 |
| CLR bit | Clears the direct bit | 2 | 3 |
| SETB C | Sets the carry flag | 1 | 1 |
| SETB bit | Sets the direct bit | 2 | 3 |
| CPL C | Complements the carry flag | 1 | 1 |
| CPL bit | Complements the direct bit | 2 | 3 |
| ANL C,bit | AND direct bit to the carry flag | 2 | 2 |
| ANL C,bit | AND complements of direct bit to the carry | 2 | 2 |
| ORL C,bit | OR direct bit to the carry flag | 2 | 2 |
| ORL C,bit | OR complements of direct bit to the carry | 2 | 2 |
| MOV C,bit | Moves the direct bit to the carry flag | 2 | 2 |
| MOV bit,C | Moves the carry flag to the direct bit | 2 | 3 |

***Table 15** • Program Branch Operations*

| Mnemonic | Description | Byte | Cycle |
|---|---|---|---|
| ACALL addr11 | Absolute subroutine call | 2 | 6 |
| LCALL addr16 | Long subroutine call | 3 | 6 |
| RET Return | Return from subroutine | 1 | 4 |
| RETI Return | Return from interrupt | 1 | 4 |
| AJMP addr11 | Absolute jump | 2 | 3 |
| LJMP addr16 | Long jump | 3 | 4 |
| SJMP rel | Short jump (relative address) | 2 | 3 |
| JMP @A + DPTR | Jump indirect relative to the DPTR | 1 | 2 |
| JZ rel | Jump if accumulator is zero | 2 | 3 |
| JNZ rel | Jump if accumulator is not zero | 2 | 3 |
| JC rel | Jump if carry flag is set | 2 | 3 |
| JNC rel | Jump if carry flag is not set | 2 | 3 |
| JB bit,rel | Jump if direct bit is set | 3 | 4 |
| JNB bit,rel | Jump if direct bit is not set | 3 | 4 |
| JBC bit,rel | Jump if direct bit is set and clears bit | 3 | 4 |
| CJNE A,direct,rel | Compares direct byte to A and jumps if not equal | 3 | 4 |
| CJNE A,#data,rel | Compares immediate to A and jumps if not equal | 3 | 4 |
| CJNE Rn,#data rel | Compares immediate to the register and jumps if not equal | 3 | 4 |
| CJNE @Ri,#data,rel | Compares immediate to indirect and jumps if not equal | 3 | 4 |
| DJNZ Rn,rel | Decrements register and jumps if not zero | 2 | 3 |
| DJNZ direct,rel | Decrements direct byte and jumps if not zero | 3 | 4 |
| NOP | No operation | 1 | 1 |

### Hexadecimal Ordered Instructions

The Core8051 instructions are listed in order of hexidecimal opcode (operation code) in Table 16.

***Table 16 •*** *Core8051 Instruction Set in Hexadecimal Order*

| Opcode | Mnemonic | Opcode | Mnemonic |
|---|---|---|---|
| 00 H | NOP | 10 H | JBC bit,rel |
| 01 H | AJMP addr11 | 11 H | ACALL addr11 |
| 02 H | LJMP addr16 | 12 H | LCALL addr16 |
| 03 H | RR A | 13 H | RRC A |
| 04 H | INC A | 14 H | DEC A |
| 05 H | INC direct | 15 H | DEC direct |
| 06 H | INC @R0 | 16 H | DEC @R0 |
| 07 H | INC @R1 | 17 H | DEC @R1 |
| 08 H | INC R0 | 18 H | DEC R0 |
| 09 H | INC R1 | 19 H | DEC R1 |
| 0A H | INC R2 | 1A H | DEC R2 |
| 0B H | INC R3 | 1B H | DEC R3 |
| 0C H | INC R4 | 1C H | DEC R4 |
| 0D H | INC R5 | 1D H | DEC R5 |
| 0E H | INC R6 | 1E H | DEC R6 |
| 0F H | INC R7 | 1F H | DEC R7 |
| 20 H | JB bit,rel | 30 H | JNB bit,rel |
| 21 H | AJMP addr11 | 31 H | ACALL addr11 |
| 22 H | RET | 32 H | RETI |
| 23 H | RL A | 33 H | RLC A |
| 24 H | ADD A,#data | 34 H | ADDC A,#data |
| 25 H | ADD A,direct | 35 H | ADDC A,direct |
| 26 H | ADD A,@R0 | 36 H | ADDC A,@R0 |
| 27 H | ADD A,@R1 | 37 H | ADDC A,@R1 |
| 28 H | ADD A,R0 | 38 H | ADDC A,R0 |
| 29 H | ADD A,R1 | 39 H | ADDC A,R1 |
| 2A H | ADD A,R2 | 3A H | ADDC A,R2 |
| 2B H | ADD A,R3 | 3B H | ADDC A,R3 |
| 2C H | ADD A,R4 | 3C H | ADDC A,R4 |
| 2D H | ADD A,R5 | 3D H | ADDC A,R5 |
| 2E H | ADD A,R6 | 3E H | ADDC A,R6 |
| 2F H | ADD A,R7 | 3F H | ADDC A,R7 |
| 40 H | JC rel | 50 H | JNC rel |
| 41 H | AJMP addr11 | 51 H | ACALL addr11 |
| 42 H | ORL direct,A | 52 H | ANL direct,A |
| 43 H | ORL direct,#data | 53 H | ANL direct,#data |
| 44 H | ORL A,#data | 54 H | ANL A,#data |
| 45 H | ORL A,direct | 55 H | ANL A,direct |
| 46 H | ORL A,@R0 | 56 H | ANL A,@R0 |
| 47 H | ORL A,@R1 | 57 H | ANL A,@R1 |

***Table 16** • Core8051 Instruction Set in Hexadecimal Order (Continued)*

| Opcode | Mnemonic | Opcode | Mnemonic |
|---|---|---|---|
| 48 H | ORL A,R0 | 58 H | ANL A,R0 |
| 49 H | ORL A,R1 | 59 H | ANL A,R1 |
| 4A H | ORL A,R2 | 5A H | ANL A,R2 |
| 4B H | ORL A,R3 | 5B H | ANL A,R3 |
| 4C H | ORL A,R4 | 5C H | ANL A,R4 |
| 4D H | ORL A,R5 | 5D H | ANL A,R5 |
| 4E H | ORL A,R6 | 5E H | ANL A,R6 |
| 4F H | ORL A,R7 | 5F H | ANL A,R7 |
| 60 H | JZ rel | 70 H | JNZ rel |
| 61 H | AJMP addr11 | 71 H | ACALL addr11 |
| 62 H | XRL direct,A | 72 H | ORL C,direct |
| 63 H | XRL direct,#data | 73 H | JMP @A+DPTR |
| 64 H | XRL A,#data | 74 H | MOV A,#data |
| 65 H | XRL A,direct | 75 H | MOV direct,#data |
| 66 H | XRL A,@R0 | 76 H | MOV @R0,#data |
| 67 H | XRL A,@R1 | 77 H | MOV @R1,#data |
| 68 H | XRL A,R0 | 78 H | MOV R0,#data |
| 69 H | XRL A,R1 | 79 H | MOV R1,#data |
| 6A H | XRL A,R2 | 7A H | MOV R2,#data |
| 6B H | XRL A,R3 | 7B H | MOV R3,#data |
| 6C H | XRL A,R4 | 7C H | MOV R4,#data |
| 6D H | XRL A,R5 | 7D H | MOV R5,#data |
| 6E H | XRL A,R6 | 7E H | MOV R6,#data |
| 6F H | XRL A,R7 | 7F H | MOV R7,#data |
| 80 H | SJMP rel | 90 H | MOV DPTR,#data16 |
| 81 H | AJMP addr11 | 91 H | ACALL addr11 |
| 82 H | ANL C,bit | 92 H | MOV bit,C |
| 83 H | MOVC A,@A+PC | 93 H | MOVC A,@A+DPTR |
| 84 H | DIV AB | 94 H | SUBB A,#data |
| 85 H | MOV direct,direct | 95 H | SUBB A,direct |
| 86 H | MOV direct,@R0 | 96 H | SUBB A,@R0 |
| 87 H | MOV direct,@R1 | 97 H | SUBB A,@R1 |
| 88 H | MOV direct,R0 | 98 H | SUBB A,R0 |
| 89 H | MOV direct,R1 | 99 H | SUBB A,R1 |
| 8A H | MOV direct,R2 | 9A H | SUBB A,R2 |
| 8B H | MOV direct,R3 | 9B H | SUBB A,R3 |
| 8C H | MOV direct,R4 | 9C H | SUBB A,R4 |
| 8D H | MOV direct,R5 | 9D H | SUBB A,R5 |
| 8E H | MOV direct,R6 | 9E H | SUBB A,R6 |
| 8F H | MOV direct,R7 | 9F H | SUBB A,R7 |
| A0 H | ORL C,bit | B0 H | ANL C,bit |
| A1 H | AJMP addr11 | B1 H | ACALL addr11 |

***Table 16 •*** *Core8051 Instruction Set in Hexadecimal Order (Continued)*

| Opcode | Mnemonic | Opcode | Mnemonic |
|---|---|---|---|
| A2 H | MOV C,bit | B2 H | CPL bit |
| A3 H | INC DPTR | B3 H | CPL C |
| A4 H | MUL AB | B4 H | CJNE A,#data,rel |
| A5 H[1] | – | B5 H | CJNE A,direct,rel |
| A6 H | MOV @R0,direct | B6 H | CJNE @R0,#data,rel |
| A7 H | MOV @R1,direct | B7 H | CJNE @R1,#data,rel |
| A8 H | MOV R0,direct | B8 H | CJNE R0,#data,rel |
| A9 H | MOV R1,direct | B9 H | CJNE R1,#data,rel |
| AA H | MOV R2,direct | BA H | CJNE R2,#data,rel |
| AB H | MOV R3,direct | BB H | CJNE R3,#data,rel |
| AC H | MOV R4,direct | BC H | CJNE R4,#data,rel |
| AD H | MOV R5,direct | BD H | CJNE R5,#data,rel |
| AE H | MOV R6,direct | BE H | CJNE R6,#data,rel |
| AF H | MOV R7,direct | BF H | CJNE R7,#data,rel |
| C0 H | PUSH direct | D0 H | POP direct |
| C1 H | AJMP addr11 | D1 H | ACALL addr11 |
| C2 H | CLR bit | D2 H | SETB bit |
| C3 H | CLR C | D3 H | SETB C |
| C4 H | SWAP A | D4 H | DA A |
| C5 H | XCH A,direct | D5 H | DJNZ direct,rel |
| C6 H | XCH A,@R0 | D6 H | XCHD A,@R0 |
| C7 H | XCH A,@R1 | D7 H | XCHD A,@R1 |
| C8 H | XCH A,R0 | D8 H | DJNZ R0,rel |
| C9 H | XCH A,R1 | D9 H | DJNZ R1,rel |
| CA H | XCH A,R2 | DA H | DJNZ R2,rel |
| CB H | XCH A,R3 | DB H | DJNZ R3,rel |
| CC H | XCH A,R4 | DC H | DJNZ R4,rel |
| CD H | XCH A,R5 | DD H | DJNZ R5,rel |
| CE H | XCH A,R6 | DE H | DJNZ R6,rel |
| CF H | XCH A,R7 | DF H | DJNZ R7,rel |
| E0 H | MOVX A,@DPTR | F0 H | MOVX @DPTR,A |
| E1 H | AJMP addr11 | F1 H | ACALL addr11 |
| E2 H | MOVX A,@R0 | F2 H | MOVX @R0,A |
| E3 H | MOVX A,@R1 | F3 H | MOVX @R1,A |
| E4 H | CLR A | F4 H | CPL A |
| E5 H | MOV A,direct | F5 H | MOV direct,A |
| E6 H | MOV A,@R0 | F6 H | MOV @R0,A |
| E7 H | MOV A,@R1 | F7 H | MOV @R1,A |
| E8 H | MOV A,R0 | F8 H | MOV R0,A |
| E9 H | MOV A,R1 | F9 H | MOV R1,A |
| EA H | MOV A,R2 | FA H | MOV R2,A |

1. *This opcode is not used by the original set of ASM51 instructions. In Core8051, this opcode is used to implement a trap instruction for the OCI debugger logic.*

***Table 16** • Core8051 Instruction Set in Hexadecimal Order (Continued)*

| Opcode | Mnemonic | Opcode | Mnemonic |
|---|---|---|---|
| EB H | MOV A,R3 | FB H | MOV R3,A |
| EC H | MOV A,R4 | FC H | MOV R4,A |
| ED H | MOV A,R5 | FD H | MOV R5,A |
| EE H | MOV A,R6 | FE H | MOV R6,A |
| EF H | MOV A,R7 | FF H | MOV R7,A |

## Instruction Definitions

All Core8051 core instructions can be condensed to 53 basic operations, alphabetically ordered according to the operation mnemonic section, as shown in Table 17.

***Table 17** • PSW Flag Modification (CY, OV, AC)*

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

***Note:*** *In this table, 'X' denotes that the indicated flag is affected by the instruction and can be a logic 1 or logic 0, depending upon specific calculations. If a particular box is blank, that flag is unaffected by the listed instruction.*

## Instruction Timing

### Program Memory Bus Cycle

The execution for instruction N is performed during the fetch of instruction N+1. A program memory fetch cycle without wait states is shown in Figure 4 on page 22. A program memory fetch cycle with wait states in shown in Figure 5 on page 22. A program memory read cycle without wait states is shown in Figure 6 on page 23. A program memory read cycle with wait states is shown in Figure 7 on page 23.

The following conventions are used in Figures 4 to 19:

***Table 18** • Conventions used in Figures 4 to 19*

| Convention | Description |
|---|---|
| Tclk | Time period of clk signal |
| N | Address of actually executed instruction |
| (N) | Instruction fetched from address N |
| N+1 | Address of next instruction |
| Addr | Address of memory cell |
| Data | Data read from address Addrl |
| read sample | Point of reading the data from the bus into the internal register |
| write sample | Point of writing the data from the bus into memory |
| ramcs | Off-core signal is made on the base ramwe and clk signals |

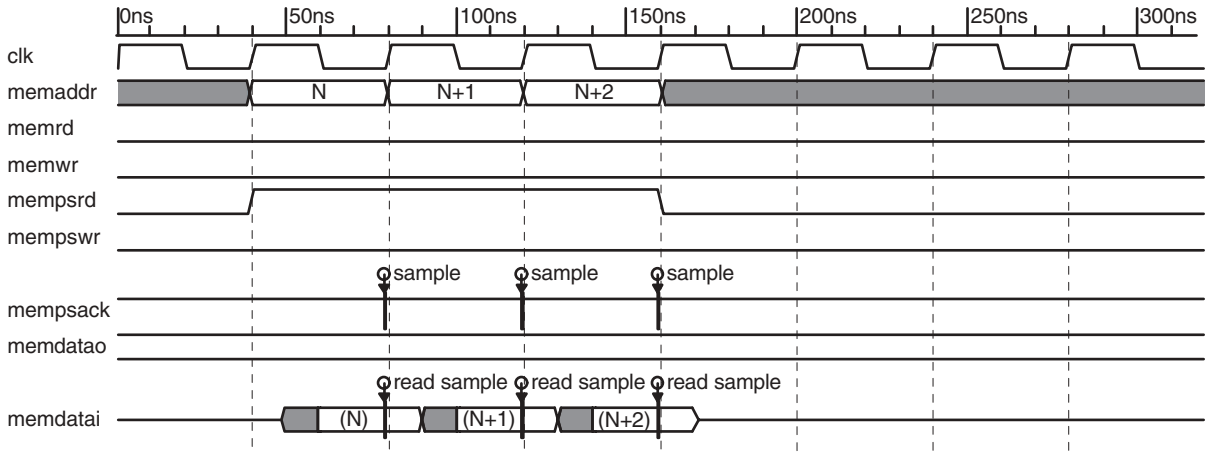


***Figure 4*** • *Program Memory Fetch Cycle without Wait States*

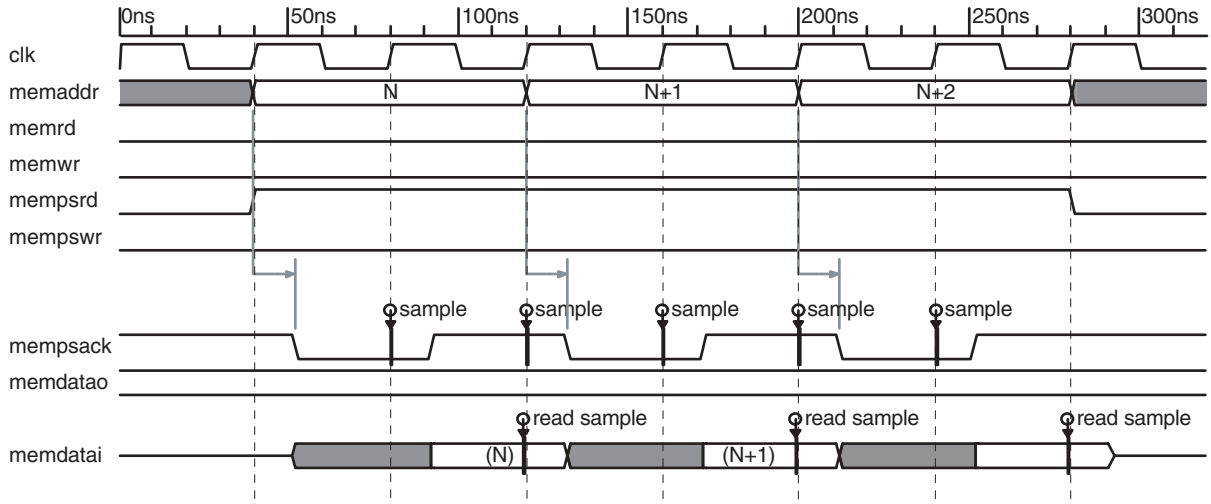


***Figure 5*** • *Program Memory Fetch with Wait States*

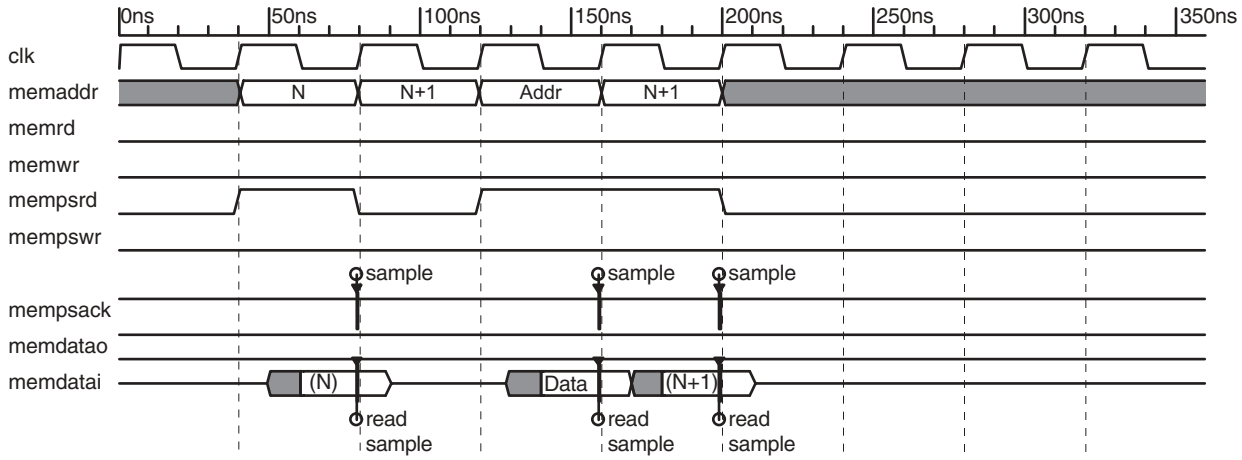


***Figure 6*** • *Program Memory Read Cycle without Wait States*

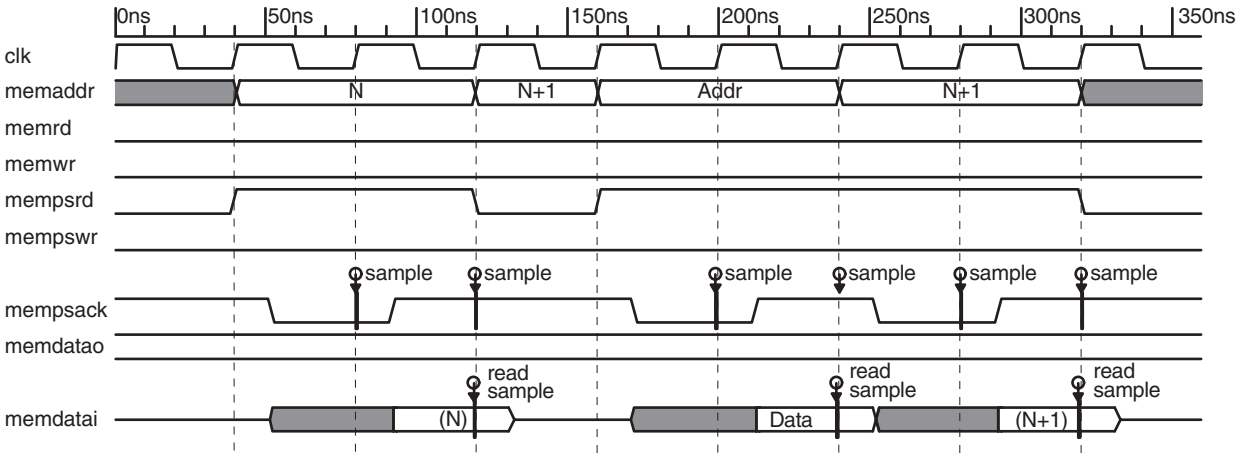


***Figure 7*** • *Program Memory Read Cycle with Wait States*

### External Data Memory Bus Cycle

Example bus cycles for external data memory access are shown in Figures 8 through 15. Figure 8 shows an external data memory read cycle without stretch cycles.

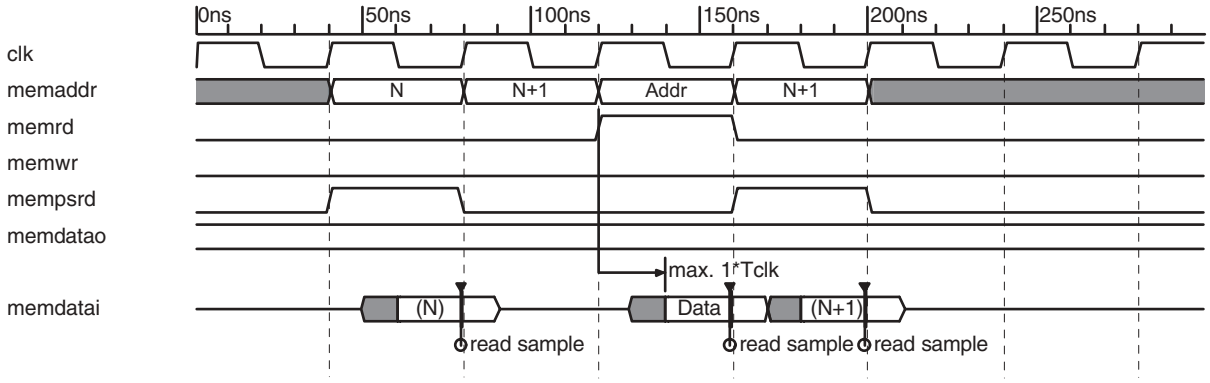


***Figure 8 •*** *External Data Memory Read Cycle without Stretch Cycles*

Figure 9 shows an external data memory read cycle with one stretch cycle.

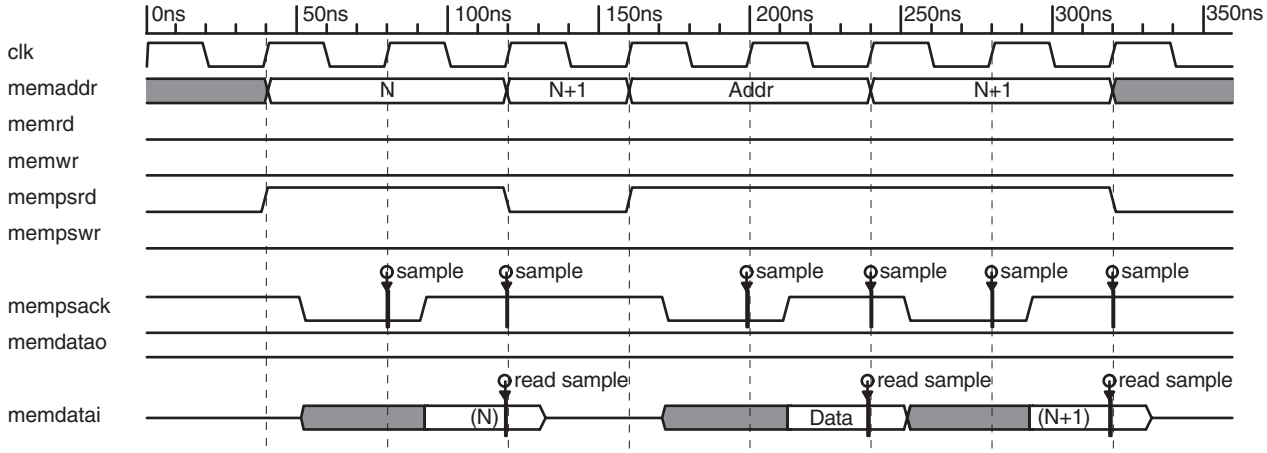


***Figure 9 •*** *External Data Memory Read Cycle with One Stretch Cycle*

Figure 10 shows an external data memory read cycle with two stretch cycles.

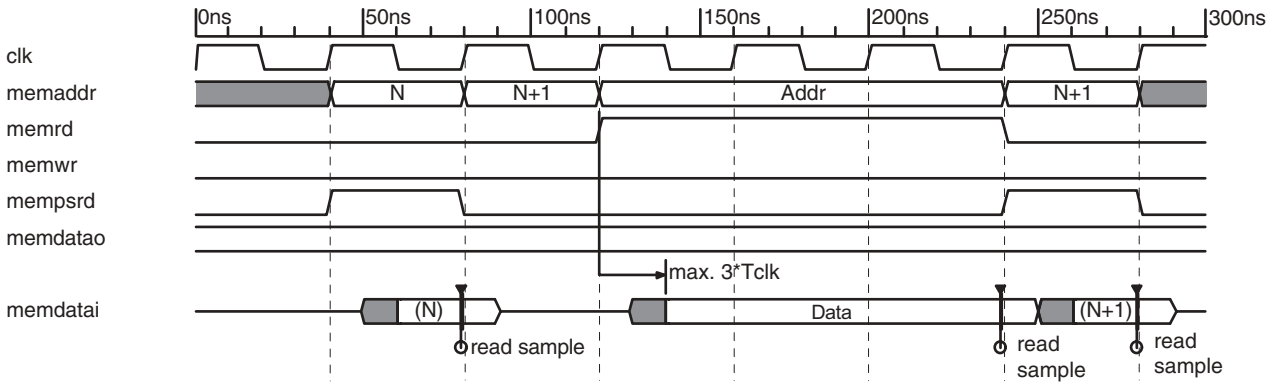


***Figure 10** • External Data Memory Read Cycle with Two Stretch Cycles*

Figure 11 shows an external data memory read cycle with seven stretch cycles.

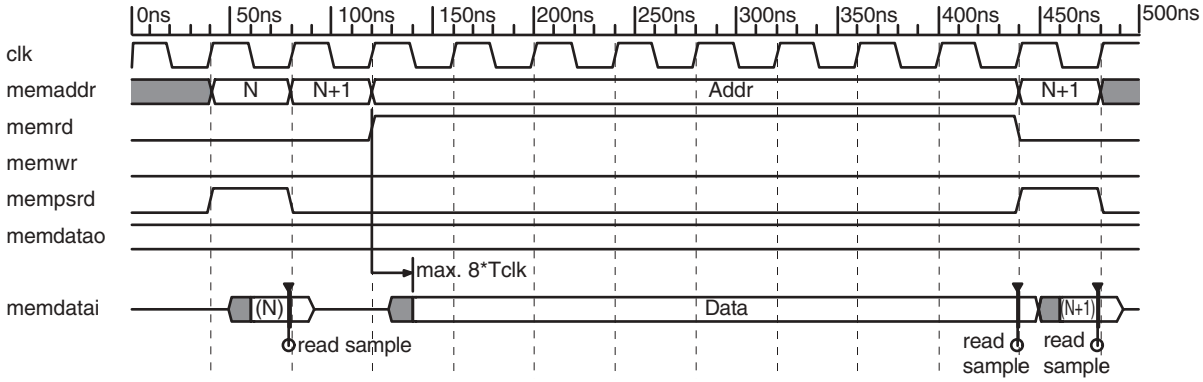


***Figure 11** • External Data Memory Read Cycle with Seven Stretch Cycles*

Figure 12 shows an external data memory write cycle without stretch cycles.

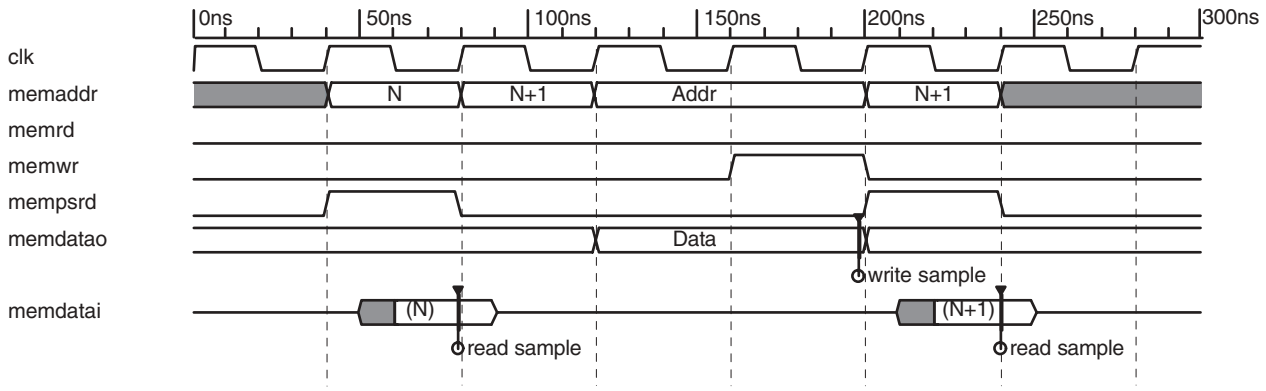


***Figure 12 •*** *External Data Memory Write Cycle without Stretch Cycles*

Figure 13 shows an external data memory write cycle with one stretch cycle.

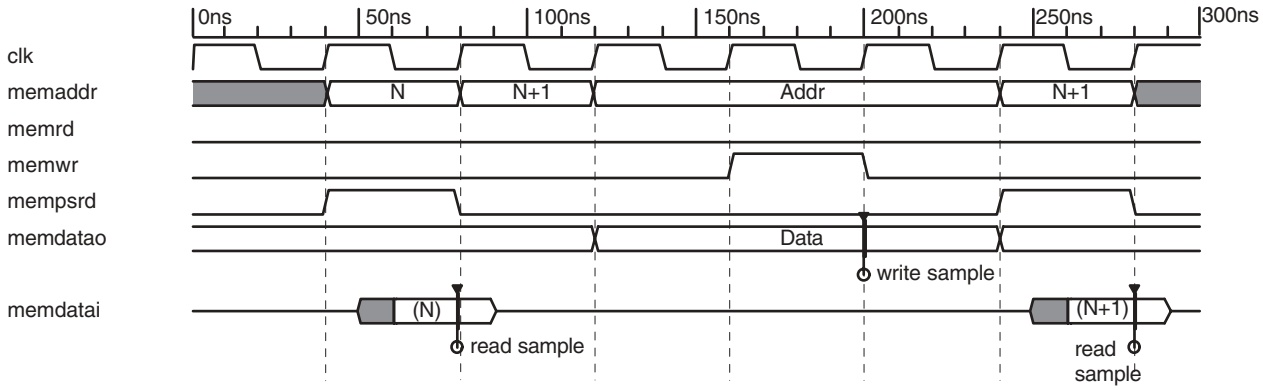


***Figure 13 •*** *External Data Memory Write Cycle with One Stretch Cycle*

Figure 14 shows an external data memory write cycle with two stretch cycles.

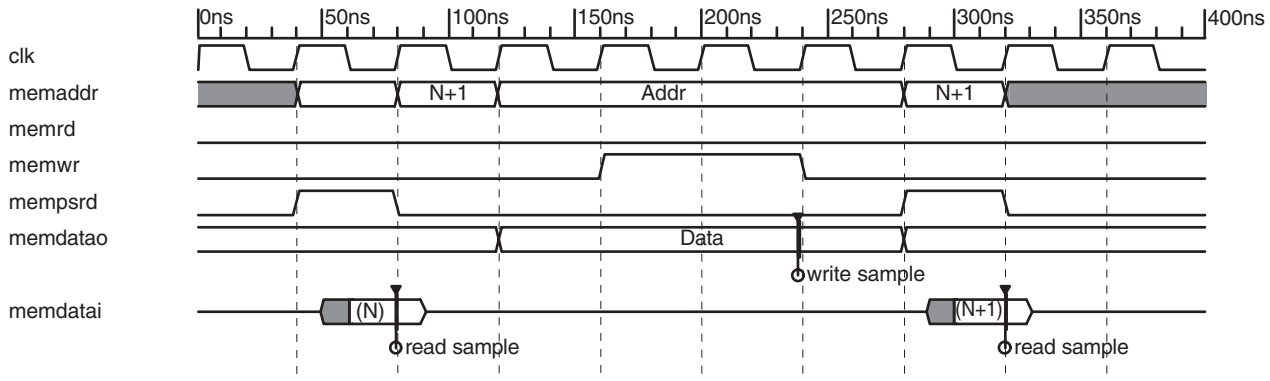


***Figure 14*** • *External Data Memory Write Cycle with Two Stretch Cycles*

Figure 15 shows an external data memory write cycle with seven stretch cycles.

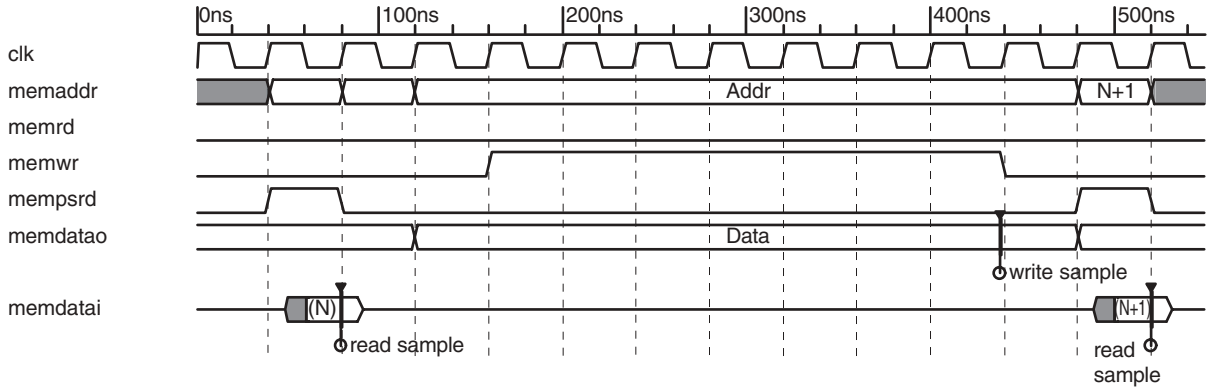


***Figure 15*** • *External Data Memory Write Cycle with Seven Stretch Cycles*

### Internal Data Memory Bus Cycle

Example bus cycles for internal data memory access are shown in Figures 16 and 17. Figure 16 shows an internal data memory read cycle.

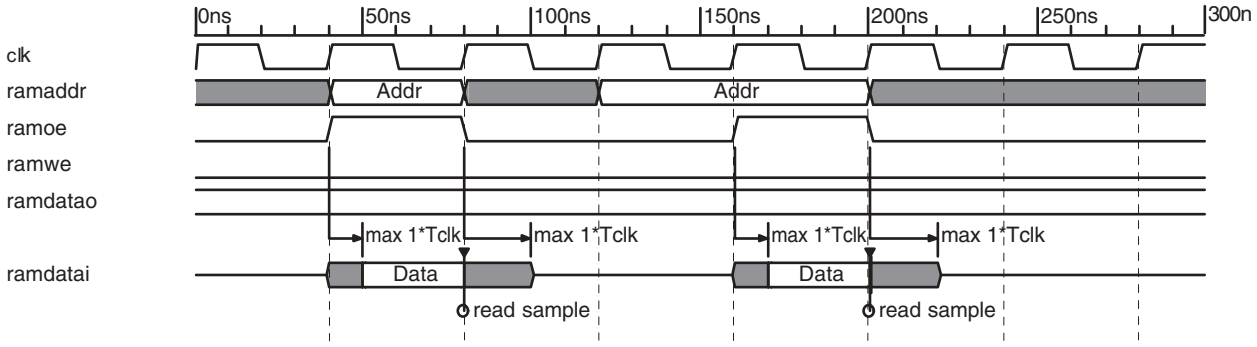


***Figure 16 •*** *Internal Data Memory Read Cycle*

Figure 17 shows an internal data memory write cycle.

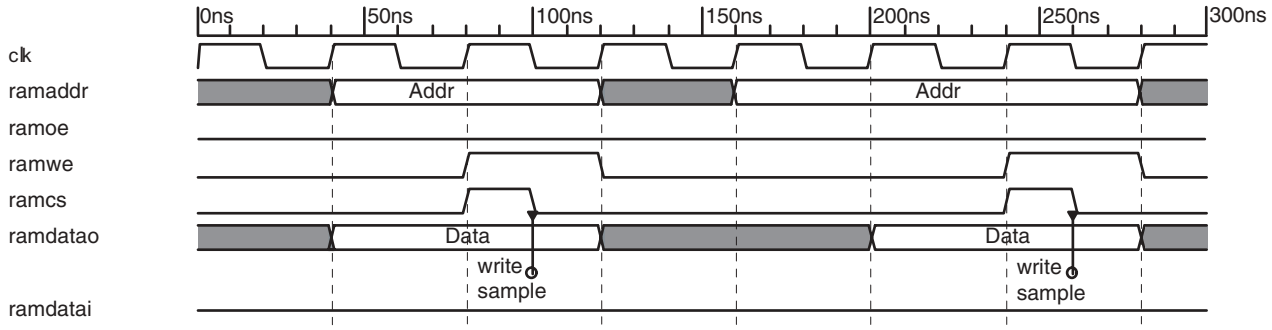


***Figure 17 •*** *Internal Data Memory Write Cycle*

### External Special Function Register Bus Cycle

Example bus cycles for external SFR access are shown in Figures 18 and 19. Figure 18 shows an external SFR read cycle. Figure 19 shows an external SFR write cycle.

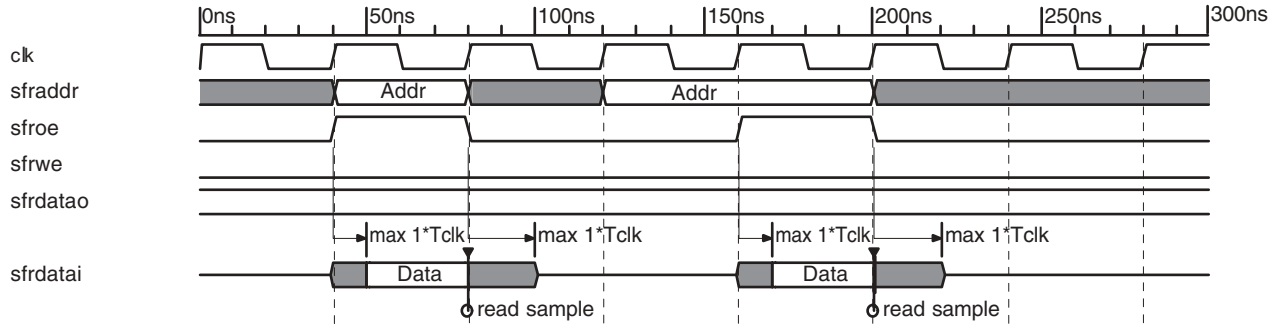


***Figure 18*** • *External SFR Read Cycle*

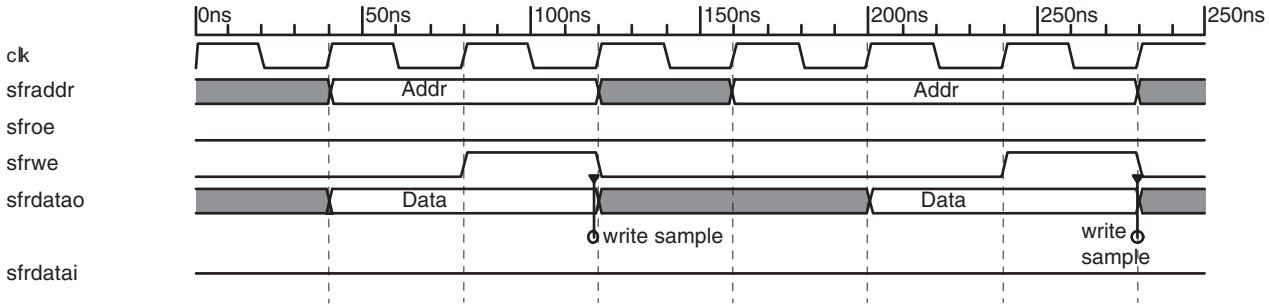


***Figure 19*** • *External SFR Write Cycle*

## Core8051 Engine

The main engine of Core8051 is composed of four components:

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

The Core8051 engine controls instruction fetches from program memory and execution using RAM or SFR. This section describes the main engine registers.

### Accumulator (acc)

The acc register is the accumulator. Most instructions use the accumulator to hold the operand. The mnemonics for accumulator-specific instructions refer to the accumulator as A, not ACC.

### B Register (b)

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

### Program Status Word (psw)

The psw register flags and bit functions are listed in Tables 19 and 20.

***Table 19 • psw Register Flags***

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| cy | ac | f0 | rs1 | rs | ov | – | p |

***Table 20 • psw Bit Functions***

| Bit | Symbol | Function |
|---|---|---|
| 7 | cy | Carry flag |
| 6 | ac | Auxiliary carry flag for BCD operations |
| 5 | f0 | General purpose flag 0 available for user |
| 4 | rs1 | Register bank select control bit 1, used to select working register bank |
| 3 | rs0 | Register bank select control bit 0, used to select working register bank |
| 2 | ov | Overflow flag |
| 1 | - | User defined flag |
| 0 | p | Parity flag, affected by hardware to indicate odd / even number of “one” bits in the accumulator, i.e. even parity |

The state of bits rs1 and rs0 from the psw register select the working registers bank as listed in Table 21.

***Table 21 • rs1/rs0 Bit Selections***

| 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 (sp)

The stack pointer is a one 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 (dptr)

The data pointer (dptr) is two bytes wide. The lower part is DPL, and the highest is DPH. It can be loaded as a two byte register (MOV DPTR,#data16) or as two registers (e.g. MOV DPL,#data8). It is generally used to access external code or data space (e.g. MOVC A,@A+DPTR or MOV A,@DPTR respectively).

### Program Counter (pc)

The program counter is two bytes wide, and is initialized to 0000H after reset. This register is incremented during fetching operation code or operation data from program memory.

### Ports

Ports p0, p1, p2, and p3 are SFRs. The contents of the SFR can be observed on corresponding pins on the chip. Writing a logic 1 to any of the ports causes the corresponding pin to be at a high level (logic 1), and writing a logic 0 causes the corresponding pin to be held at a low level (logic 0).

All four ports on the chip are bidirectional. Each bit of each port consists of a register, an output driver, and an input buffer. Core8051 can output or read data through any of these ports if they are not used for alternate purposes.

## Timers/Counters

### Timers 0 and 1

Core8051 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 cycle, which means that it counts up after every 12 oscillator periods.

In counter mode, the register is incremented when a falling edge is observed at the corresponding t0 or t1 input pin. Since it takes two machine cycles to recognize a logic 1 to logic 0 transition event, the maximum input count rate is 1/24 of the oscillator (clk input pin) frequency. There are no restrictions on the duty cycle. However, an input should be stable for at least one machine cycle (12 clock periods) to ensure proper recognition of a logic 0 or logic 1 value.

Four operating modes can be selected for Timer 0 and Timer 1. Two SFRs (tmod and tcon) are used to select the appropriate mode. The various register flags, bit descriptions, and mode descriptions are listed in Tables 22 to 24.

### Timer/Counter Mode Control Register (tmod)

Table 22 displays the tmod register functions.

***Table 22** • tmod Register Flags*

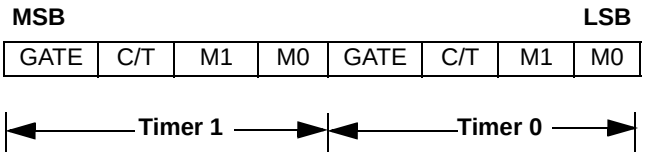


Table 23 provides tmod register bits descriptions.

***Table 23** • tmod Register Bits Description*

| Bit | Symbol | Function |
|---|---|---|
| 7,3 | GATE | If set, enables external gate control (pin int0 or int1 for Counter 0 or Counter 1, respectively). When int0 or int1 is high, and the trx bit is set (see tcon register), the counter is incremented every falling edge on the t0 or t1 input pin |
| 6, 2 | C/T | Selects Timer or Counter operation. When set to logic 1, a Counter operation is performed. When cleared to logic 0, the corresponding register will function as a Timer |
| 5, 1 | M1 | Selects the mode for Timer/Counter 0 or Timer/Counter 1 |
| 4, 0 | M0 | Selects the mode for Timer/Counter 0 or Timer/Counter 1 |

Table 24 provides timer and counter mode descriptions.

***Table 24** • Timers/Counter Mode Description*

| M1 | M0 | Mode | Function |
|---|---|---|---|
| 0 | 0 | Mode 0 | 13-bit Counter/Timer, with five lower bits in the tl0 or tl1 register and eight bits in the th0 or th1 register (for Timer 0 and Timer 1, respectively). The three high order bits of the tl0 and tl1 registers are held at zero |
| 0 | 1 | Mode 1 | 16-bit Counter/Timer |
| 1 | 0 | Mode2 | Eight-bit auto-reload Counter/Timer. The reload value is kept in the th0 or th1 register, while the tl0 or tl1 register is incremented every machine cycle. When the tl0 or tl1 register overflows, the value in the th0 or th1 register is copied to the tl0 or tl1 register, respectively |
| 1 | 1 | Mode3 | If the M1 and M0 bits in Timer 1 are set to logic 1, Timer 1 stops. If the M1 and M0 bits in Timer 0 are set to logic 1, Timer 0 acts as two independent eight-bit Timers/Counters. |

***Note:*** *The th0 register is affected by the tr1 bit in the tcon register. When the th0 register overflows, the tf1 flag in the tcon register is set.*

### Timer/Counter Control Register (tcon)

Table 25 displays the tcon register flags.

***Table 25 •*** *tcon Register Flags*

| MSB | | | | | | | LSB |
|---|---|---|---|---|---|---|---|
| TF1 | TR1 | TF0 | TR0 | IE1 | IT1 | IE0 | IT0 |

Table 26 displays the tcon register bit functions.

***Table 26 •*** *tcon Register Bit Functions*

| Bit | Symbol | Function |
|---|---|---|
| 7 | TF1 | Timer 1 overflow flag. This flag is set when Timer 1 overflows. This flag should be cleared by the user's software |
| 6 | TR1 | Timer 1 Run control bit. If cleared, Timer 1 stops |
| 5 | TF0 | Timer 0 overflow flag. This flag is set when Timer 0 overflows. This flag should be cleared by the user's software |
| 4 | TR0 | Timer 0 Run control bit. If cleared, Timer 0 stops |
| 3 | IE1 | Interrupt 1 edge flag. This flag is set when a falling edge on the external pin int1 is observed. This flag is cleared when an interrupt is processed |
| 2 | IT1 | Interrupt 1 type control bit. This bit selects whether a falling edge or a low level on input pin int1 causes an interrupt |
| 1 | IE0 | Interrupt 0 edge flag. This flag is set when a falling edge on the external pin int0 is observed. This flag is cleared when an interrupt is processed |
| 0 | IT0 | Interrupt 0 type control bit. This bit selects whether a falling edge or a low level on input pin int0 causes an interrupt |

## Serial Interface

### Serial Port 0

The serial buffer consists of two separate registers: transmit buffer and receive buffer. Writing data to the SFR sbuf sets this data in serial output buffer and starts the transmission. Reading from the sbuf register reads data from the serial receive buffer.

The serial port can simultaneously transmit and receive data. It can also buffer one 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 one of four modes.

**Mode 0**

In this mode, the rxd0i pin receives serial data and the rxd0o pin transmits serial data. The txd0 pin outputs the shift clock. Eight bits are transmitted with LSB first. The baud rate is fixed at 1/12 of the crystal (clk input) frequency.

**Mode 1**

In this mode, the rxd0i pin receives serial data and the txd0 pin transmits serial data. No external shift clock is used, and the following 10 bits are transmitted:

- A Start Bit (always 0)
- Eight Data Bits (LSB first)
- A Stop Bit (always 1)

On receive, a start bit synchronizes the transmission, eight data bits are available by reading the sbuf register, and a stop bit sets the flag RB8 in the SFR scon.

**Mode 2**

This mode is similar to Mode 1 but has two main differences. The baud rate is fixed at 1/32 or 1/64 of the oscillator (clk input) frequency, and the following 11 bits are transmitted or received:

- A Start Bit (0)
- Eight Data Bits (LSB first)
- A Programmable Ninth Bit
- A Stop Bit (1)

The ninth bit can be used to control the parity of the serial interface. At transmission, the TB8 bit in the scon register is output as the ninth bit, and at receive, the ninth bit affects the RB8 bit in the SFR 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 RI flag in the scon register to logic 0 and the REN flag in the scon register to logic 1. In other modes, if the REN flag is a logic 1, the reception of serial data will begin with a start bit.

### Multiprocessor Communication

The nine bit reception feature in Modes 2 and 3 can be used for multiprocessor communication. In this case, the SM2 bit in the scon register is set to logic 1 by the slave processors. When the master processor outputs the slave address, it sets the ninth bit to logic 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 this message. After addressing the slave, the master will output the rest of the message with the 9th bit set to logic 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 setting of the Serial Port Control Register scon. The various register flags, bit descriptions, mode descriptions, and baud rates are listed in Tables 27 to 30.

Note that in the following tables, “fosc” represents the frequency of the clk input signal.

***Table 27** • scon Register Flags*

| **MSB** | | | | | | | **LSB** |
|---|---|---|---|---|---|---|---|
| SM0 | SM1 | SM2 | REN | TB8 | RB8 | TI | RI |

***Table 28** • scon Bit Functions*

| Bit | Symbol | Function |
|---|---|---|
| 7 | SM0 | Sets baud rate |
| 6 | SM1 | Sets baud rate |
| 5 | SM2 | Enables multiprocessor communication feature |
| 4 | REN | If set, enables serial reception. Cleared by software to disable reception |
| 3 | TB8 | The ninth 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.) |
| 2 | RB8 | In Modes 2 and 3 it is the ninth 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 the software |
| 1 | TI | Transmits the interrupt flag and is set by the hardware after completion of a serial transfer. Must be cleared by the software |
| 0 | RI | Receives the interrupt flag and is set by the hardware after completion of a serial reception. Must be cleared by the software |

***Table 29 •*** *Serial Port Modes*

| SM0 | SM1 | Mode | Description | Baud Rate |
|---|---|---|---|---|
| 0 | 0 | 0 | Shift register | fosc/12 |
| 0 | 1 | 1 | Eight-bit UART | variable |
| 1 | 0 | 2 | Nine-bit UART | fosc/32 or /64 |
| 1 | 1 | 3 | Nine-bit UART | variable |

***Table 30 •*** *Serial Port Baud Rates*

| Mode | Baud rate |
|---|---|
| Mode 0 | fosc12 |
| Mode 1,3 | Timer 1 overflow rate |
| Mode 2 | SMOD = 0 fosc/64<br>SMOD = 1 fosc/32 |

#### 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 at auto in auto-reload mode to establish a baud rate, the following equation is useful:

$$\text{Baud Rate} = \frac{2^{\text{SMOD}} \times \text{fosc}}{32 \times 12 \times (256 - \text{th1})}$$

## Interrupt Service Routine Unit

Core8051 provides 13 interrupt sources with four priority levels. Each source has its own request flag(s) located in a SFR (tcon, ircon, scon). Each interrupt requested by the corresponding flag can be individually enabled or disabled by the enable bits in the ien0 and ien1 registers. There are two external interrupts accessible through pins int0 and int1: edge or level sensitive (falling edge or low 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 (int0 and int1) interrupt level or transition activity is made by setting the IT1 and IT0 bits in the SFR tcon.

When the interrupt event happens, a corresponding interrupt control bit is set in the tcon register (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 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 (logic 0 or logic 1).

During high to low transitions, recognition of an interrupt event is possible if both high and low levels last at least one machine cycle.

#### Timer 0 and Timer 1 Interrupts

Timer 0 and 1 interrupts are generated by the TF0 and TF1 flags in the tcon register, 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 if Core8051 has accessed the corresponding interrupt service vector. This can only be done if this interrupt is enabled in the ien0 register.

#### Serial Port Interrupt

The serial port interrupt is generated by logical OR of the TI and RI flags in the SFR scon. The TI flag is set after the data transmission completes. The RI flag is set when the last bit of the incoming serial data was read. Neither RI nor TI is cleared by Core8051, so the user's interrupt service routine must clear these flags.

#### Special Function Registers

Table 31 displays the Interrupt Enable 0 register (ie0).

***Table 31 •*** *ien0 Register*

| MSB | | | | | | | LSB |
|---|---|---|---|---|---|---|---|
| eal | – | – | es0 | et1 | ex1 | et0 | ex0 |

Table 32 provides the ien0 bit functions.

***Table 32 •*** *ien0 Bit Functions*

| Bit | Symbol | Function |
|---|---|---|
| 7 | eal | eal=0 – disable all interrupts |
| 6 | – | Not used for interrupt control |
| 5 | – | Not used for interrupt control |
| 4 | es0 | es0=0 – disable serial channel 0 interrupt |
| 3 | et1 | et1=0 – disable timer 1 overflow interrupt |
| 2 | ex1 | ex1=0 – disable external interrupt 1 |
| 1 | et0 | et0=0 – disable timer 0 overflow interrupt |
| 0 | ex0 | ex0=0 – disable external interrupt 0 |

Table 33 displays the Interrupt Enable 1 register (ien1).

***Table 33 •*** *ien1 Register*

| MSB | | | | | | | LSB |
|---|---|---|---|---|---|---|---|
| ex7 | ex6 | ex5 | ex4 | ex3 | ex2 | ex1 | ex0 |

Table 34 provides the ien1 bit functions.

***Table 34** • ien1 Bit Functions*

| Bit | Symbol | Function |
|---|---|---|
| 7 | ex7 | ex7=0 – disable int7 |
| 6 | ex6 | ex6=0 – disable int6 |
| 5 | ex5 | ex5=0 – disable int5 |
| 4 | ex4 | ex4=0 – disable int4 |
| 3 | ex3 | ex3=0 – disable int3 |
| 2 | ex2 | ex2=0 – disable int2 |
| 1 | ex1 | ex1=0 – disable int1a |
| 0 | ex0 | ex0=0 – disable int0a |

### Priority Level Structure

All interrupt sources are combined in priority level groups and controlled in terms of priority level by bits in the ip0 and ip1 registers.

Table 35 displays the Interrupt Priority 0 register (ip0).

***Table 35** • ip0 Register*

| MSB | | | | | | | LSB |
|---|---|---|---|---|---|---|---|
| – | – | ip0.5 | ip0.4 | ip0.3 | ip0.2 | ip0.1 | ip0.0 |

Table 36 displays the Interrupt Priority 1 register (ip1).

***Table 36** • ip1 Register*

| MSB | | | | | | | LSB |
|---|---|---|---|---|---|---|---|
| – | – | ip1.5 | ip1.4 | ip1.3 | ip1.2 | ip1.1 | ip1.0 |

Each group of interrupt sources can be programmed individually to one of four priority levels by setting or clearing one bit in the special function register ip0 and one in ip1. If requests of the same priority level are received simultaneously, an internal polling sequence determines which request is serviced first.

For example, in Table 38 the two interrupts, Timer 0 interrupt and external pin int2, are combined in a priority group and are priority level controlled by the combination of bit0 from the ip0 register and bit0 from the ip1 register.

Table 37 displays the priority levels.

***Table 37** • Priority Levels*

| ip1.x | ip0.x | Priority Level |
|---|---|---|
| 0 | 0 | Level0 (lowest) |
| 0 | 1 | Level1 |
| 1 | 0 | Level2 |
| 1 | 1 | Level3 (highest) |

Table 38 displays the groups of priority.

***Table 38** • Groups of Priority*

| Bit | Group |
|---|---|
| ip1.0,ip0.0 | External interrupt 0(ie0), int0a, int1a |
| ip1.1,ip0.1 | Timer 0 interrupt, int2 |
| ip1.2,ip0.2 | External interrupt 1(ie1), int3 |
| ip1.3,ip0.3 | Timer 1 interrupt, int4 |
| ip1.4,ip0.4 | Serial channel 0 interrupt, int5 |
| ip1.5,ip0.5 | int6, int7 |

Table 39 displays the polling sequence.

***Table 39** • Polling Sequence*

| |
|---|
| External interrupt 0(ie0) |
| int0a |
| int1a |
| Timer 0 interrupt |
| int2 |
| External interrupt 1(ie1) |
| int3 |
| Timer 1 interrupt |
| int4 |
| Serial channel 0 interrupt |
| int5 |
| int6 |
| int7 |

### Interrupt Vectors

The interrupt vector addresses are listed in Table 40.

***Table 40 • Interrupt Vector Addresses***

| Interrupt Request Flags | Interrupt Vector Address |
|---|---|
| ie0 – External interrupt 0 | 0003H |
| tf0 – Timer 0 interrupt | 000BH |
| ie1 – External interrupt 1 | 0013H |
| tf1 – Timer 1 interrupt | 001BH |
| ri0/ti0 – Serial channel 0 interrupt | 0023H |
| int6 | 002BH |
| int0a | 0083H |
| int1a | 0043H |
| int2 | 004BH |
| int3 | 0053H |
| int4 | 005BH |
| int5 | 0063H |
| int7 | 006BH |

### Interrupt Detect

The interrupts int0a, int1a, and int2 to int7 are activated by level and the active state is logic 1 (high). Each of these interrupt pins must be held at a logic 1 value until Core8051 starts to service the affected interrupt. The user's software must take the appropriate action to clear each interrupt request (by writing to external peripherals via the external SFR interface).

### External Interrupt Connection (int)

Table 41 displays the interrupt source connection.

***Table 41 • Interrupt Source Connection***

| Interrupt | Device | Source |
|---|---|---|
| ie0 | – | External pin |
| ie1 | – | External pin |
| Tf0 | Timer 0 | – |
| Tf1 | Timer 1 | – |
| Ri0 | Serial 0 | – |
| Ti0 | Serial 0 | – |
| int0a | – | External pin |
| int1a | – | External pin |
| int2 | – | External pin |
| int3 | – | External pin |
| int4 | – | External pin |
| int5 | – | External pin |
| int6 | – | External pin |
| int7 | – | External pin |

Figure 20 on page 37 illustrates an overview of the interrupt service routine hardware within Core8051, including the polling sequence.

## ISR Structure

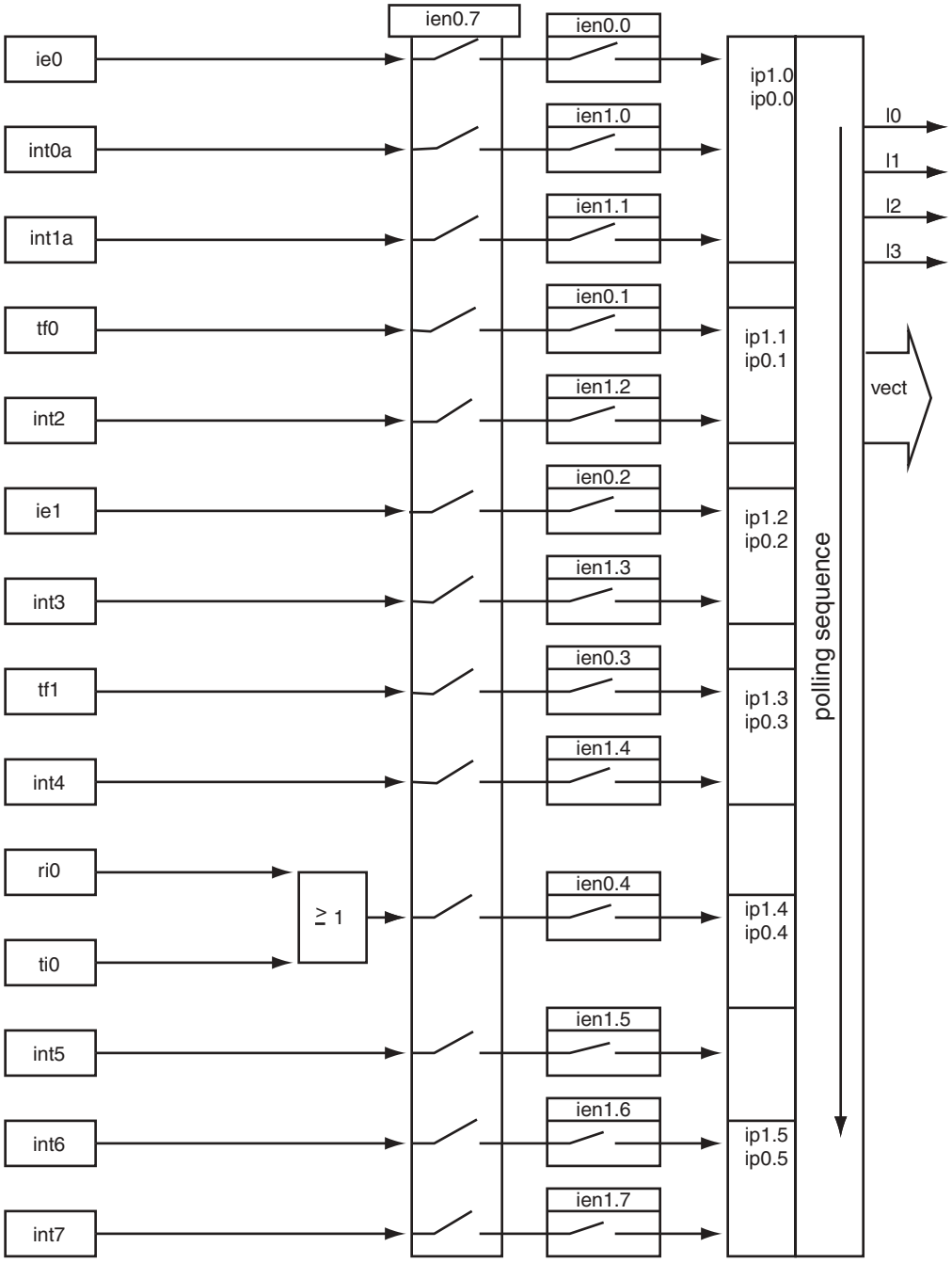


***Figure 20*** • *ISR Structure*

## Power Management Unit

The Power Management Unit monitors two power management modes: IDLE and STOP.

### Idle Mode

Setting the idle bit of the pcon register invokes the IDLE mode. The IDLE mode leaves internal clocks and peripherals running. Power consumption drops because the CPU is not active. The CPU can exit the IDLE state with any interrupts or a reset.

### Stop Mode

Setting the stop bit of the pcon register invokes the STOP mode. All internal clocking in this mode is turned off. The CPU will exit this state from a non-clocked external interrupt or a reset condition. Internally generated interrupts (timer, serial port, etc.) are not useful since they require clocking activity.

### Special Function Registers

Table 42 displays the pcon register.

***Table 42 •*** *pcon Register*

| MSB | | | | | | | LSB |
|---|---|---|---|---|---|---|---|
| smod | – | – | – | gf1 | gf0 | stop | idle |

Table 43 provides the pcon bit functions.

***Table 43 •*** *pcon Bit Functions*

| Bit | Symbol | Function |
|---|---|---|
| 7 | smod | Not used for power management |
| 6 | – | – |
| 5 | – | – |
| 6 | – | – |
| 3 | gf1 | General purpose flag 1 |
| 4 | gf0 | General purpose flag 0 |
| 1 | stop | Stop mode control bit. Setting this bit places Core8051 into Stop Mode. This bit is always read as logic 0 |
| 0 | idle | Idle mode control bit. Setting this bit places Core8051 into Idle Mode. This bit is always read as logic 0 |

## Power Management Block Diagram

The power management /clock gating logic is illustrated in Figure 21.

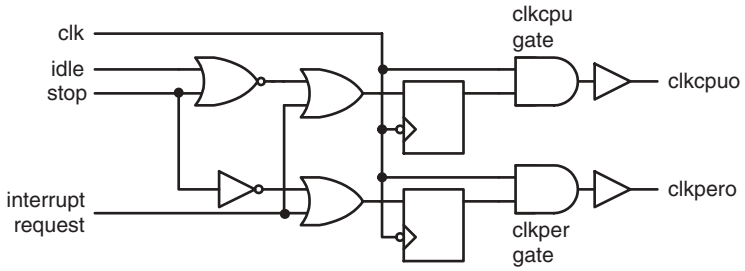


***Figure 21 •*** *Power Management Block Diagram*

The clkcpuo and clkpero outputs are globally buffered and can be used to connect to external peripherals (within an Actel FPGA). The clkpero output is suggested over the clkcpuo output for connection to peripherals, as it will be active during the Idle mode.

The falling edge of the clk input signal is used to synchronize the drivers for gating the two clock signals: clkcpuo and clkpero.

## Interface for On-Chip Instrumentation (Optional)

The optional OCI unit serves as the interface for on-chip instrumentation. The OCI communicates with external debugger hardware and software as a debugging aid to the user.

The following signals are not directly visible at the I/O pins of Core8051, they are connected internally between the OCI block and the main logic of Core8051:

- debugreq
- debugack
- debugstep
- debugprog
- fetch
- flush
- instr
- acc

RTL licensees of Core8051 have access to these internal signals. The JTAG interface pins: TCK, TMS, TDI, TDO, and nTRT are used in conjunction with external debugger hardware and software to control and monitor the above-mentioned OCI signals.

### The Run/Stop Control

The debugger controls the CPU with the debugreq signal. The debugreq signal stops the CPU at the next instruction and holds it in an idle state. When in an idle state, the CPU executes the NOP instruction and returns the debugack signal. The debugreq signal is synchronized to the microcontroller instruction cycle and phase, as shown in Figure 22. Figure 22 demonstrates the behavior of the debugreq and debugack signals. The LCALL and the LJMP are sample instructions of a user program.

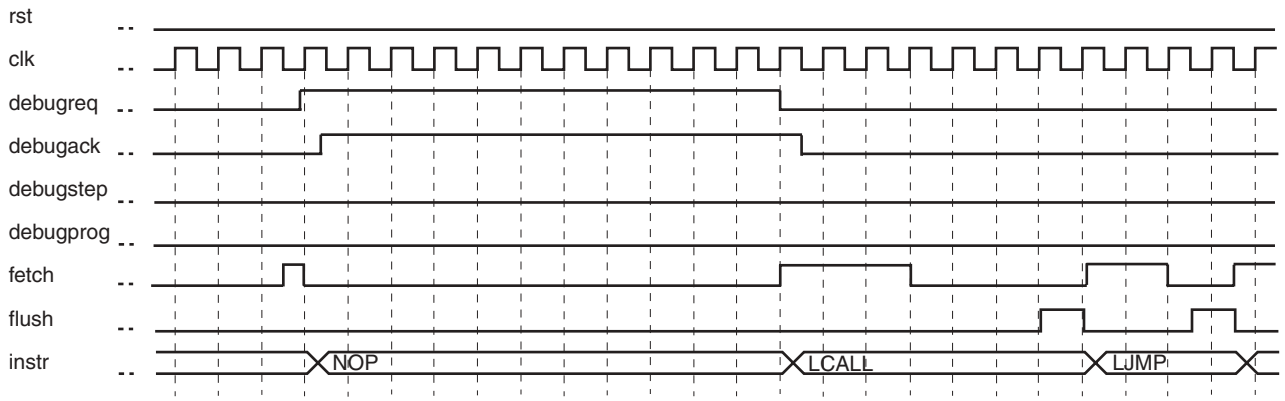


***Figure 22** • The Run/Stop Control*

### Single-Step Mode

To execute one instruction in the debug mode, the OCI asserts a signal debugstep for one system clock. The CPU responds by negating debugack, executing one user or debugger instruction, and then asserting debugack. The OCI can set debugprog high for execution of a debugger instruction or set debugprog low for execution of a user instruction (see Figure 23).

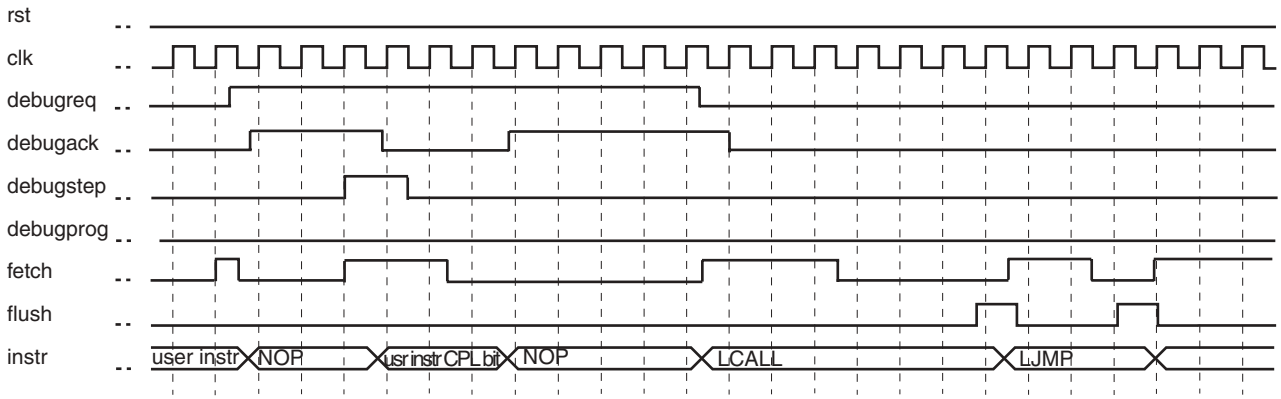


***Figure 23 •*** *Single-Step Mode*

### Software breakpoint

When the CPU executes the opcode 0xA5, the core enters debug mode and asserts the debugack signal. The debugger responds by setting debugreq high. The CPU leaves the debug mode when debugreq is high for at least one clock period and then goes low.

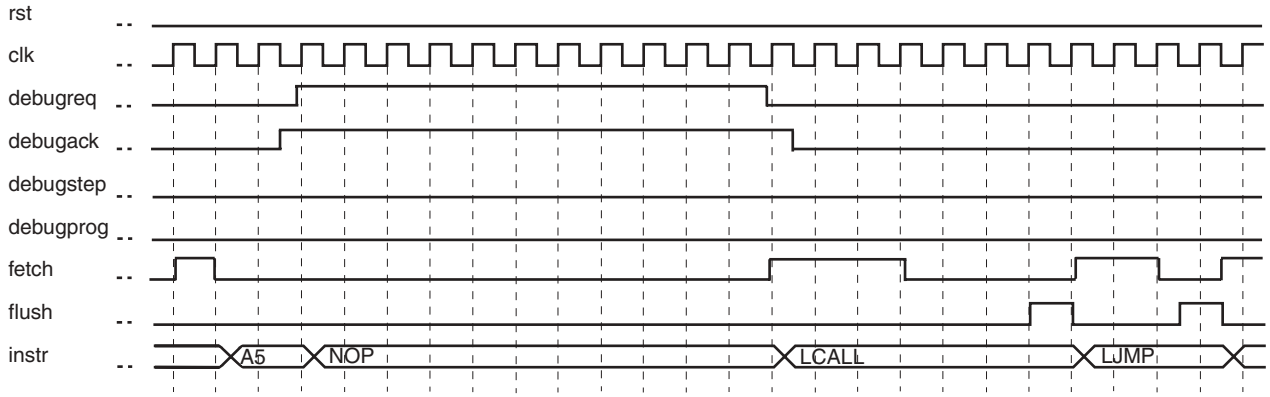


***Figure 24 •*** *Software Breakpoint*

### Debugger Program

The debugger can submit an instruction to the CPU. The OCI logic uses a multiplexer on the program memory input bus (memdatai) to optionally override the user instruction with the debugger instruction.

The interrupts are disabled during execution of the debugger program. Setting the interrupt flag does not cause an interrupt request.

The power down IDLE and STOP modes are supported. The CPU can only exit the IDLE or STOP state with a reset.

### Hardware Breakpoint

The debugger can monitor the program memory address bus (memaddr) for optional hardware breakpoint addresses. When a fetch is noted from this address, the debugger can replace the original user instruction with opcode 0xA5.

When the CPU executes the opcode 0xA5, the core enters the debug mode and asserts the debugack signal. The debugger responds by setting debugreq high.

The OCI can monitor the external memory address and data buses (memaddr, memdatai, memdatao) to monitor the

program execution. The buses to and from internal data memory (ramaddr, ramdatao, ramdatai) are also visible for monitoring.

### Program Trace

Core8051 provides several signals for tracing program execution. Two signals, fetch and flush, are internally connected to the OCI logic to monitor instruction fetch activity. The fetch signal is active when Core8051 performs an instruction fetch, and the flush signal is active when Core8051 fetches the first instruction after a branch instruction.

The following signals are used to connect Core8051 to optional external RAM devices for debug mode trace memory control: TraceA, TraceDI, TraceDO, and TraceWr. Example wrapper RTL source code is provided with the RTL and Netlist releases of Core8051 to illustrate the connection of the ProASIC$^{PLUS}$ and Axcelerator RAM cells that are used as optional trace memory.

### Access to ACC (Accumulator) Register

The external debugger hardware and software can observe the contents of the ACC register by way of the optional OCI logic block (through the JTAG interface).

## Ordering Information

Order Core8051 through your local Actel sales representative. Use the following number convention when ordering: Core8051-XX, where XX is listed in Table 44.

***Table 44 •*** *Ordering Information*

| XX | Description |
|---|---|
| EV | Evaluation Version |
| SN | Single-use Netlist for use on Actel devices |
| AN | Netlist for unlimited use on Actel devices |
| AR | RTL for unlimited use on Actel devices |
| UR | RTL for unlimited use and not restricted to Actel devices |

http://www.actel.com

**Actel Corporation**
955 East Arques Avenue
Sunnyvale, California 94086
USA
**Tel:** (408) 739-1010
**Fax:** (408) 739-1540

**Actel Europe Ltd.**
Dunlop House, Riverside Way
Camberley, Surrey GU15 3YL
United Kingdom
**Tel:** +44 (0)1276 401450
**Fax:** +44 (0)1276 401490

**Actel Japan**
EXOS Ebisu Bldg. 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150 Japan
**Tel:** +81 03-3445-7671
**Fax:** +81 03-3445-7668

**Actel Hong Kong**
39th Floor
One Pacific Place
88 Queensway
Admiralty, Hong Kong
Tel: 852-22735712

51700022-0/6.03