
Core8051 Instruction Set Details

User's Guide



Actel Corporation, Mountain View, CA 94043

© 2003 Actel Corporation. All rights reserved.

Printed in the United States of America

Part Number: 50200005-1

Release: October 2003

No part of this document may be copied or reproduced in any form or by any means without prior written consent of Actel.

Actel makes no warranties with respect to this documentation and disclaims any implied warranties of merchantability or fitness for a particular purpose. Information in this document is subject to change without notice. Actel assumes no responsibility for any errors that may appear in this document.

This document contains confidential proprietary information that is not to be disclosed to any unauthorized person without prior written consent of Actel Corporation.

Trademarks

Actel and the Actel logo are registered trademarks of Actel Corporation.

Adobe and Acrobat Reader are registered trademarks of Adobe Systems, Inc.

All other products or brand names mentioned are trademarks or registered trademarks of their respective holders.

Table of Contents

| | |
|---|----|
| Introduction 1 | |
| Document Organization | 1 |
| Actel Datasheet | 1 |
| 1 Mnemonic Notes 3 | |
| 2 Instruction List - Functions 5 | |
| 3 Instruction List - Hexadecimal Code 11 | |
| 4 Instructions 17 | |
| ACALL addr11 | 17 |
| ADD A, <src-byte> | 18 |
| ADDC A, <src-byte> | 20 |
| AJMP addr11 | 22 |
| ANL <dest-byte>, <src-byte> | 22 |
| ANL C, <src-bit> | 25 |
| CJNE <dest-byte>, <src-byte>, rel | 26 |
| CLR A | 29 |
| CLR <bit-type> | 29 |
| CPL A | 30 |
| CPL <bit-type> | 31 |
| DA A | 32 |
| DEC byte | 33 |
| DIV AB | 35 |
| DJNZ <byte>, <rel-addr> | 36 |
| INC <byte>. | 37 |
| INC DPTR | 39 |
| JB bit, rel | 40 |
| JBC bit,rel | 40 |
| JC rel | 41 |
| JMP @A + DPTR | 42 |
| JNB bit,rel | 42 |

| | |
|--|----|
| JNC rel | 43 |
| JNZ rel | 44 |
| JZ rel | 45 |
| LCALL addr16 | 45 |
| LJMP addr16. | 46 |
| MOV <dest-byte>, <src-byte> | 47 |
| MOV <dest-bit>, <src-bit>. | 53 |
| MOV DPTR, #data16. | 54 |
| MOVC A, @A + <base-reg>. | 55 |
| MOVX <dest-byte>, <src-byte> | 56 |
| MUL AB. | 58 |
| NOP | 59 |
| ORL <dest-byte>, <src-byte> | 59 |
| ORL C, <src-bit> | 62 |
| POP direct | 63 |
| PUSH direct | 64 |
| RET | 64 |
| RETI. | 65 |
| RL A | 66 |
| RLC A | 66 |
| RR A | 67 |
| RRC A | 68 |
| SETB <bit-type>. | 68 |
| SJMP rel | 69 |
| SUBB A, <src-byte> | 70 |
| SWAP A | 72 |
| XCH A, <byte> | 73 |
| XCHD A,@Ri | 74 |
| XRL <dest-byte>, <src-byte>. | 75 |

Index 79

Introduction

This guide contains detailed information about all of the instructions supported by Core8051. A brief example of how each instruction might be used is given, as well as its effect on the Program Status Word (PSW) flags from the ALU. The number of bytes, the binary machine language encoding, and a symbolic description or restatement of the function is also provided.

Note: 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 that alter directly addressed registers could affect the other status flags if the instruction is applied to the PSW. Status flags can also be modified by bit manipulation.

Document Organization

This guide is divided into the following chapters:

[Chapter 1 – Mnemonic Notes](#) details the mnemonic notes used throughout this document.

[Chapter 2 – Instruction List - Functions](#) provides a concise list of the instructions, grouped according to function.

[Chapter 3 – Instruction List - Hexadecimal Code](#) for a list of the instructions, listed in order of hexadecimal opcode.

[Chapter 4 – Instructions](#) provides detailed information about all of the instructions supported by Core8051.

Actel Datasheet

Datasheets are available on our web site at <http://www.actel.com/techdocs/ds/>.

For Core8051 specific information, refer to the Core8051 datasheet.

Mnemonic Notes

This chapter details the mnemonic notes used in this document.

Table 1-1. Notes on Data Addressing Modules

| | |
|----------|---|
| Rn | Working register R0-R7. Listed in Encoding sections as “r r r” |
| 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. Listed in Encoding sections as “i” |
| #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 1-2. 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 +127/-128 bytes relative to the first byte of the following instruction |

Table 1-3. Additional Notation

| | |
|------|---------------------|
| PC | Program Counter |
| SP | Stack Pointer |
| PSW | Program Status Word |
| C | Carry Flag |
| DPTR | Data Pointer |
| P | Parity Flag |

Table 1-4. Operators

| | |
|----|----------------------------------|
| + | Bit-wise addition |
| - | Bit-wise subtraction |
| / | Unsigned bit-wise division |
| * | Unsigned bit-wise multiplication |
| | Bit-wise logical OR |
| & | Bit-wise logical AND |
| ~ | Bit-wise logical NOT |
| ^ | Bit-wise logical XOR |
| | Boolean logical OR |
| && | Boolean logical AND |
| ! | Boolean logical NOT |
| < | Less than |
| > | Greater than |
| <> | Not equal |
| = | Equal |

Instruction List - Functions

This chapter provides a concise list of the instructions used in this document. The instructions are grouped according to function.

Table 2-1. 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 |

Table 2-1. Arithmetic Operations (Continued)

| | | | |
|------------|-----------------------------|---|---|
| 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 2-2. 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 |

Table 2-2. Logic Operations (Continued)

| | | | |
|------------------|---|---|---|
| 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 2-3. 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 |

Table 2-3. Data Transfer Operations (Continued)

| | | | |
|-------------------|---|---|------|
| 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 |

Table 2-3. Data Transfer Operations (Continued)

| | | | |
|------------|--|---|---|
| XCHD A,@Ri | Exchanges the low-order nibble indirect RAM with A | 1 | 3 |
|------------|--|---|---|

Table 2-4. 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 2-5. 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 |

Table 2-5. Program Branch Operations (Continued)

| | | | |
|--------------------|---|---|---|
| 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 |

Instruction List - Hexadecimal Code

This chapter lists the instructions in order of hexadecimal code.

Table 3-1. Core8051 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 |

Table 3-1. Core8051 Instruction Set in Hexadecimal Order (Continued)

| Opcode | Mnemonic | Opcode | Mnemonic |
|--------|------------------|--------|------------------|
| 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 |
| 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 |

Table 3-1. Core8051 Instruction Set in Hexadecimal Order (Continued)

| Opcode | Mnemonic | Opcode | Mnemonic |
|--------|------------------|--------|------------------|
| 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,direct |
| 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 |
| 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 |

Table 3-1. Core8051 Instruction Set in Hexadecimal Order (Continued)

| Opcode | Mnemonic | Opcode | Mnemonic |
|--------|-------------------|--------|-------------------|
| 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 |

Table 3-1. Core8051 Instruction Set in Hexadecimal Order (Continued)

| Opcode | Mnemonic | Opcode | Mnemonic |
|--------|----------------|--------|--------------------|
| 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 |
| 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 |

Table 3-1. Core8051 Instruction Set in Hexadecimal Order (Continued)

| Opcode | Mnemonic | Opcode | Mnemonic |
|--------|--------------|--------|--------------|
| 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 |

Instructions

This chapter lists the Core8051 instructions in alphabetical order.

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, operation code bits 7, 6, and 5, and the second byte of the instruction. The subroutine called must therefore start within the same 2kB 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)) ← (PC[7:0])
(SP) ← (SP) + 1
((SP)) ← (PC[15:8])
(PC[10:0]) ← 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 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 of bit 7, or a carry out of bit 7 but not 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

$$(A) \leftarrow (A) + (Rn)$$

Bytes

1

Encoding

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

ADD A, direct

Operation

ADD

$$(A) \leftarrow (A) + (\text{direct})$$

ADD A, @Ri

ADD A, <src-byte>

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | direct address |
|---|---|---|---|---|---|---|---|----------------|

ADD A, @Ri

Operation

ADD

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

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

ADD A, #data

Operation

ADD

$$(A) \leftarrow (A) + \#data$$

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 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 of bit 7, or a carry out of bit 7 but not 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

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

Bytes

1

Encoding

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

ADDC A, direct

Operation

ADDC

$$(A) \leftarrow (A) + (C) + (\text{direct})$$

ADDC A, @Ri

ADDC A, <src-byte>

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | direct address |
|---|---|---|---|---|---|---|---|----------------|

ADDC A, @Ri

Operation

ADDC

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

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

ADDC A, #data

Operation

ADDC

$$(A) \leftarrow (A) + (C) + \#data$$

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), operation code bits 7, 6, and 5, and the second byte of the instruction. The destination must be within the same 2kB block of program memory as the first byte of the instruction following AJMP.

Operation

AJMP

$$(PC) \leftarrow (PC) + 2$$

$$(PC[10:0]) \leftarrow \text{page address}$$

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

Bit-wise 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 an 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

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

ANL A,Rn

Operation

ANL

$$(A) \leftarrow (A) \& (Rn)$$

Bytes

1

Encoding

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

ANL A,direct

Operation

ANL

$$(A) \leftarrow (A) \& (\text{direct})$$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | direct address |
|---|---|---|---|---|---|---|---|----------------|

ANL A, @Ri

Operation

ANL

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

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

ANL A, #data

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

ANL A, #data

Operation

ANL

$(A) \leftarrow (A) \& \#data$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | immediate data |
|---|---|---|---|---|---|---|---|----------------|

ANL direct,A

Operation

ANL

$(direct) \leftarrow (direct) \& (A)$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | direct address |
|---|---|---|---|---|---|---|---|----------------|

ANL direct, #data

Operation

ANL

$$(\text{direct}) \leftarrow (\text{direct}) \ \& \ \#data$$

Bytes

3

Encoding

| | | | | | | | |
|----------------|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| direct address | | | | | | | |
| immediate data | | | | | | | |

ANL C, <src-bit>

Function

Bit-wise logical AND for bit variables

Description

If the Boolean value of the source bit is a logic 0, then the carry flag is cleared. Otherwise, the carry flag is left in its current state. A tilde (“~”) preceding the operand in the assembly language indicates that the bit-wise 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

$$(C) \leftarrow (C) \ \& \ (bit)$$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|-------------|
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | bit address |
|---|---|---|---|---|---|---|---|-------------|

ANL C,~bit**Operation**

ANL

 $(C) \leftarrow (C) \& \sim (\text{bit})$ **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 and 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

```
(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

```
(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

```

(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

```

(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) $\leftarrow 0$

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

CLR <bit-type>

Function

Clear bit

Description

The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.

CLR C

Operation

CLR C

(C) $\leftarrow 0$

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

CLR bit**Operation**

CLR

 $(\text{bit}) \leftarrow 0$ **Bytes**

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|-------------|
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | bit address |
|---|---|---|---|---|---|---|---|-------------|

CPL A**Function**

Complement accumulator

Description

Each bit of the accumulator is complemented (one's complement). Bits that previously contained a logic 1 are changed to logic 0 and vice versa. No flags are affected.

Operation

CPL

 $(A) \leftarrow \sim (A)$

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

CPL <bit-type>

Function

Complement bit

Description

The bit variable specified is complemented. A bit that was a one is changed to zero and vice versa. No other flags are affected. CPL can operate on the carry or any directly addressable bit.

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

CPL C

Operation

CPL

 $(C) \leftarrow \sim (C)$ **Bytes**

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

CPL bit

Operation

CPL

 $(bit) \leftarrow \sim (bit)$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|-------------|
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | bit address |
|---|---|---|---|---|---|---|---|-------------|

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 be used to perform the addition. If accumulator bits 3 down to 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 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 one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the accumulator, depending on initial accumulator and PSW 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[3:0]) > 9) || ((AC) = 1))
then A[3:0] ← (A[3:0]) + 6
and
if ((A[7:4]) > 9) || ((C) = 1))
then (A[7:4]) ← (A[7:4]) + 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 is read from the output data latch, not the input pins.

DEC A

Operation

DEC

$$(A) \leftarrow (A) - 1$$

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

DEC Rn**Operation**

DEC

$$(Rn) \leftarrow (Rn) - 1$$

Bytes

1

Encoding

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

DEC direct**Operation**

DEC

$$(\text{direct}) \leftarrow (\text{direct}) - 1$$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | direct address |
|---|---|---|---|---|---|---|---|----------------|

DEC @Ri

Operation

DEC

$$((Ri)) \leftarrow ((Ri)) - 1$$

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 while 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 are undefined and the overflow flag is set. The carry flag is cleared in any case.

Operation

DIV

$$\begin{aligned} (A) &\leftarrow \text{quotient } ((A) / (B)) \\ (B) &\leftarrow \text{remainder } ((A) / (B)) \end{aligned}$$

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 is 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 is read from the output data latch, not the input pins.

DJNZ Rn,rel

Operation

DJNZ

$$\begin{aligned} (PC) &\leftarrow (PC) + 2 \\ (Rn) &\leftarrow (Rn) - 1 \\ \text{if } (Rn) > 0 \text{ or } (Rn) < 0 \\ \text{then } (PC) &\leftarrow (PC) + \text{rel} \end{aligned}$$
Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|--------------|
| 1 | 1 | 0 | 1 | 1 | r | r | r | rel. address |
|---|---|---|---|---|---|---|---|--------------|

DJNZ direct,rel

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. Three addressing modes are allowed: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data is 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

$$(Rn) \leftarrow (Rn) + 1$$

Bytes

1

Encoding

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

INC direct**Operation**

INC

$$(\text{direct}) \leftarrow (\text{direct}) + 1$$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | direct address |
|---|---|---|---|---|---|---|---|----------------|

INC @Ri

Operation

INC

$$((Ri)) \leftarrow ((Ri)) + 1$$

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

INC DPTR

Function

Increment data pointer

Description

Increments 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 that can be incremented.

Operation

INC

$$(DPTR) \leftarrow (DPTR) + 1$$

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, jumps to the address indicated. Otherwise, proceeds 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 |
|---|---|---|---|---|---|---|---|

JBC bit,rel

Function

Jump if bit is set and clear bit

Description

If the indicated bit is one, branches to the address indicated. Otherwise, proceeds with the next instruction. In either case, the designated bit is cleared. 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 is 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 is set

Description

If the carry flag is set, branches to the address indicated. Otherwise, proceeds 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

Description

Add the eight-bit unsigned contents of the accumulator with the sixteen-bit data pointer and loads the resulting sum into the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2^{16}) and 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

$$(PC) \leftarrow (A) + (DPTR)$$

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, branches to the indicated address. Otherwise, proceeds 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 is not set

Description

If the carry flag is a zero, branches to the address indicated. Otherwise, proceeds 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, branches to the indicated address. Otherwise, proceeds 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) <> 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, branches to the address indicated. Otherwise, proceeds 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 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) ← addr[15:0]
```

Bytes

3

Encoding

| | | | | | | | |
|------------|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| addr[15:8] | | | | | | | |
| addr[7:0] | | | | | | | |

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) ← addr[15:0]
```

MOV A,Rn

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

Bytes

3

Encoding

| | | | | | | | |
|-------------|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| addr [15:8] | | | | | | | |
| addr [7:0] | | | | | | | |

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

$(A) \leftarrow (Rn)$

Bytes

1

Encoding

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

MOV A,direct¹

Operation

MOV

$(A) \leftarrow (\text{direct})$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | direct address |
|---|---|---|---|---|---|---|---|----------------|

MOV A,@Ri

Operation

MOV

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

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

MOV A, #data

Operation

MOV

$(A) \leftarrow \#data$

1. MOV A,ACC is not a valid instruction. The content of the accumulator after the execution of this instruction is undefined.

MOV Rn,A

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

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | immediate data |
|---|---|---|---|---|---|---|---|----------------|

MOV Rn,A

Operation

MOV

$(Rn) \leftarrow (A)$

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

MOV Rn,direct

Operation

MOV

$(Rn) \leftarrow (\text{direct})$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 1 | 0 | 1 | 0 | 1 | r | r | r | direct address |
|---|---|---|---|---|---|---|---|----------------|

MOV Rn, #data

Operation

MOV

$(Rn) \leftarrow \#data$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 0 | 1 | 1 | 1 | 1 | r | r | r | immediate data |
|---|---|---|---|---|---|---|---|----------------|

MOV direct,A

Operation

MOV

$(direct) \leftarrow (A)$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | direct address |
|---|---|---|---|---|---|---|---|----------------|

MOV direct,Rn

Operation

MOV

$(direct) \leftarrow (Rn)$

MOV direct,direct

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

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 1 | 0 | 0 | 0 | 1 | r | r | r | direct address |
|---|---|---|---|---|---|---|---|----------------|

MOV direct,direct

Operation

MOV

$(\text{direct}) \leftarrow (\text{direct})$

Bytes

3

Encoding

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

MOV direct, @ Ri

Operation

MOV

$(\text{direct}) \leftarrow ((\text{Ri}))$

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 (source) | | | | | | | |
| immediate data | | | | | | | |

MOV @ Ri,A

Operation

MOV

((Ri)) ← (A)

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

MOV @ Ri,direct

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

MOV @ Ri,direct

Operation

MOV

$((Ri)) \leftarrow (direct)$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | i | direct address |
|---|---|---|---|---|---|---|---|----------------|

MOV @ Ri,#data

Operation

MOV

$((Ri)) \leftarrow \#data$

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

$(C) \leftarrow (bit)$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|-------------|
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | bit address |
|---|---|---|---|---|---|---|---|-------------|

MOV bit,C

Operation

MOV

$(bit) \leftarrow (C)$

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.

MOVC A, @A + DPTR

MOVC A, @A + <base-reg>

Operation

MOV

$(DPTR) \leftarrow \#data[15:0]$
 $DPH \ DPL \leftarrow \#data[15:8] \ \#data[7:0]$

Bytes

3

Encoding

| | | | | | | | |
|-----------------------|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 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 16-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. 16-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

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

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

MOVC A, @A + PC

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

MOVC A, @A + PC

Operation

MOVC

$(PC) \leftarrow (PC) + 1$
 $(A) \leftarrow ((A) + (PC))$

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 eight-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 eight-bit address. In the second type of MOVX instructions, the data pointer generates a 16-bit address. Please refer to the section “External Data Memory” of the Core8051 data sheet. This data sheet can be found at <http://www.actel.com/techdocs/ds/>.

MOVX A,@Ri

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

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 eight-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 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

```
product ← (A) * (B)
(A) ← product[7:0], (B) ← product[15:8]
if (product > 255)
then OV ← 1
else OV ← 0
```

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

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

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

Function

Bit-wise 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, 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 is read from the output data latch, not the input pins.

ORL A,Rn

Operation

ORL

$$(A) \leftarrow (A) \mid (Rn)$$

Bytes

1

Encoding

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

ORL A,direct

Operation

ORL

$$(A) \leftarrow (A) \mid (\text{direct})$$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | direct address |
|---|---|---|---|---|---|---|---|----------------|

ORL A,@Ri

Operation

ORL

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

ORL A,#data

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

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

ORL A,#data

Operation

ORL

$(A) \leftarrow (A) \mid \#data$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | immediate data |
|---|---|---|---|---|---|---|---|----------------|

ORL direct,A

Operation

ORL

$(direct) \leftarrow (direct) \mid (A)$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | direct address |
|---|---|---|---|---|---|---|---|----------------|

ORL direct, #data

Operation

ORL

$(\text{direct}) \leftarrow (\text{direct}) \mid \#data$

Bytes

3

Encoding

| | | | | | | | |
|----------------|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| direct address | | | | | | | |
| Immediate data | | | | | | | |

ORL C, <src-bit>

Function

Bit-wise logical OR for bit variables

Description

Sets the carry flag if the Boolean value is a logic 1. Leaves the carry in its current state otherwise. A tilde (“~”) preceding the operand in the assembly language indicates that the bit-wise 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

$(C) \leftarrow (C) \mid (\text{bit})$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|-------------|
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | bit address |
|---|---|---|---|---|---|---|---|-------------|

ORL C,_{~bit}

Operation

ORL

$$(C) \leftarrow (C) \mid \sim (\text{bit})$$

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 transferred to the directly addressed byte indicated. No flags are affected.

Operation

POP

$$\begin{aligned} (\text{direct}) &\leftarrow ((SP)) \\ (SP) &\leftarrow (SP) - 1 \end{aligned}$$

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

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

$$((SP)) \leftarrow (direct)$$
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 instruction. No flags are affected.

Operation

RET

```
(PC[15:8]) ← ((SP))  
(SP) ← (SP) - 1  
(PC[7:0]) ← ((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 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 is executed before the pending interrupt is processed.

Operation

RETI

```
(PC[15:8]) ← ((SP))  
(SP) ← (SP) - 1  
(PC[7:0]) ← ((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

$$(A[7:1]) \leftarrow (A[6:0])$$

$$(A[0]) \leftarrow (A[7])$$
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 rotated together one bit to the left. Bit 7 moves into the carry flag and the original state of the carry flag moves into the bit 0 position. No other flags are affected.

Operation

RLC

$$\begin{aligned}(A[7:1]) &\leftarrow (A[6:0]) \\ (A[0]) &\leftarrow (C) \\ (C) &\leftarrow (A[7])\end{aligned}$$

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

$$\begin{aligned}(A[6:0]) &\leftarrow (A[7:1]) \\ (A[7]) &\leftarrow (A[0])\end{aligned}$$

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 rotated together one bit to the right. Bit 0 moves into the carry flag and the original value of the carry flag moves into the bit 7 position. No other flags are affected.

Operation

RRC

$$\begin{aligned}(A[6:0]) &\leftarrow (A[7:1]) \\ (A[7]) &\leftarrow (C) \\ (C) &\leftarrow (A[0])\end{aligned}$$

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

SETB <bit-type>

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 C

Operation

SETB

$(C) \leftarrow 1$

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

SETB bit

Operation

SETB

$(\text{bit}) \leftarrow 1$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|-------------|
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | bit address |
|---|---|---|---|---|---|---|---|-------------|

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, a SJMP with a displacement of 0FEH would be a one-instruction infinite loop.

Operation

SJMP

$$\begin{aligned}(\text{PC}) &\leftarrow (\text{PC}) + 2 \\(\text{PC}) &\leftarrow (\text{PC}) + \text{rel}\end{aligned}$$

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 from the accumulator, leaving the result in the accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, otherwise it clears C. (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 is set if a borrow is needed for bit 3, and cleared otherwise. OV 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

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

Bytes

1

Encoding

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

SUBB A,direct

Operation

SUBB

$$(A) \leftarrow (A) - (C) - (\text{direct})$$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | direct address |
|---|---|---|---|---|---|---|---|----------------|

SUBB A, @ Ri

Operation

SUBB

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

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

SUBB A, #data**Operation**

SUBB

$$(A) \leftarrow (A) - (C) - \#data$$

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. The operation can also be thought of as a four-bit rotate instruction. No flags are affected.

Operation

SWAP

$A[3:0] \leftarrow A[7:4], A[7:4] \leftarrow A[3:0]$

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, while writing the original accumulator contents to the indicated variable at the same time. The source/destination operand can use register, direct or register-indirect addressing.

XCH A,Rn

Operation

XCH

$(A) \leftarrow (Rn), (Rn) \leftarrow (A)$

Bytes

1

Encoding

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

XCH A,direct

Operation

XCH

$$(A) \leftarrow (\text{direct}), (\text{direct}) \leftarrow (A)$$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | direct address |
|---|---|---|---|---|---|---|---|----------------|

XCH A, @ Ri

Operation

XCH

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

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 (generally representing a hexadecimal or BCD digit), with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles of each register are not affected. No flags are affected.

XRL A,Rn

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

Operation

XCHD

$$(A[3:0]) \leftarrow ((Ri[3:0])), ((Ri[3:0])) \leftarrow (A[3:0])$$

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

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

Function

Bit-wise 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, 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 accumulator or immediate data.

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

XRL A,Rn

Operation

XRL

$$(A) \leftarrow (A) \wedge (Rn)$$

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

XRL A,direct

Bytes

1

Encoding

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

XRL A,direct

Operation

XRL

$(A) \leftarrow (A) \wedge (\text{direct})$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | direct address |
|---|---|---|---|---|---|---|---|----------------|

XRL A, @ Ri

Operation

XRL

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

Bytes

1

Encoding

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

XRL A, #data

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

XRL A, #data

Operation

XRL

$(A) \leftarrow (A) \wedge \#data$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | immediate data |
|---|---|---|---|---|---|---|---|----------------|

XRL direct,A

Operation

XRL

$(direct) \leftarrow (direct) \wedge (A)$

Bytes

2

Encoding

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----------------|
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | direct address |
|---|---|---|---|---|---|---|---|----------------|

XRL direct, #data

Operation

XRL

$$(\text{direct}) \leftarrow (\text{direct}) \wedge \#data$$

Bytes

3

Encoding

| | | | | | | | |
|----------------|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| direct address | | | | | | | |
| immediate data | | | | | | | |

Index

A

ACALL addr11 17
Actel
 Manuals 1
ADD A 18
ADD A, #data 19
ADD A, @Ri 19
ADD A, direct 18
ADD A, Rn 18
ADDC A 20
ADDC A, #data 21
ADDC A, @Ri 21
ADDC A, direct 20
ADDC A, Rn 20
AJMP addr11 22
ANL 22
ANL A, #data 24
ANL A, @Ri 23
ANL A,direct 23
ANL A,Rn 23
ANL C 25
ANL C,~bit 26
ANL C,bit 25
ANL direct, #data 25
ANL direct,A 24

C

CJNE 26
CJNE @Ri, #data, rel 28
CJNE A, #data,rel 27
CJNE A,direct,rel 27
CJNE RN, #data, rel 28
CLR A 29
CLR bit 29, 30
CPL A 30
CPL bit 31

CPL C 31

D

DA A 32
DEC @Ri 35
DEC A 33
DEC byte 33
DEC direct 34
DEC Rn 34
DIV AB 35
DJNZ 36
DJNZ direct,rel 37
DJNZ Rn,rel 36
Document Organization 1

I

INC 37
INC @Ri 39
INC A 37
INC direct 38
INC DPTR 39
INC Rn 38

J

JB bit, rel 40
JBC bit,rel 40
JC rel 41
JMP @A + DPTR 42
JNB bit,rel 42
JNC rel 43
JNZ rel 44
JZ rel 45

L

LCALL addr16 45
LJMP addr16 46

M

Mnemonic 3
MOV 47, 53
MOV @ Ri, #data 53
MOV @ Ri, A 52
MOV @ Ri, direct 53
MOV A, #data 48
MOV A, @Ri 48
MOV A, direct *) 48
MOV A, Rn 47
MOV bit, C 54
MOV C, bit 54
MOV direct, #data 52
MOV direct, @ Ri 51
MOV direct, A 50
MOV direct, direct 51
MOV direct, Rn 50
MOV DPTR, #data16 54
MOV Rn, #data 50
MOV Rn, A 49
MOV Rn, direct 49
MOVC A, @A 55
MOVC A, @A + DPTR 55
MOVC A, @A + PC 56
MOVX 56
MOVX @DPTR, A 58
MOVX @Ri, A 57
MOVX A, @DPTR 57
MOVX A, @Ri 57
MUL AB 58

N

NOP 59

O

Ordering Information 78

ORL 59
ORL A, #data 61
ORL A, @Ri 60
ORL A, direct 60
ORL A, Rn 60
ORL C 62
ORL C, ~bit 63
ORL C, bit 62
ORL direct, #data 62
ORL direct, A 61

P

POP direct 63
PUSH direct 64

R

Related Manuals 1
RET 64
RETI 65
RL A 66
RLC A 66
RR A 67
RRC A 68

S

SETB 68
SETB bit 69
SETB C 69
SJMP rel 69
SUBB A 70
SUBB A, #data 72
SUBB A, @ Ri 71
SUBB A, direct 71
SUBB A, Rn 71
SWAP A 72

X

XCH A 73

XCH A, @ Ri 74

XCH A,direct 74

XCH A,Rn 73

XCHD A,@Ri 74

XRL 75

XRL A, #data 77

XRL A, @ Ri 76

XRL A,direct 76

XRL A,Rn 75

XRL direct, #data 78

XRL direct,A 77

***For more information about Actel's products, visit our website at
<http://www.actel.com>***

Actel Corporation • 2061 Stierlin Court • Mountain View, CA 94043 USA
Customer Service: 650.318.1010 • Customer Applications Center: 800.262.1060

Actel Europe Ltd. • Dunlop House, Riverside Way • Camberley, Surrey GU15 3YL United Kingdom
Tel: +44 (0)1276.401452 • Fax: +44 (0)1276.401490

Actel Japan • EXOS Ebisu Bldg. 4F • 1-24-14 Ebisu Shibuya-ku • Toyko 150 • Japan
Tel: +81.03.3445.7671 Fax: +81.03.3445.7668

50200005-1/10.03

