DP80390

Pipelined High Performance Microcontroller

Instructions set details

ver 3.10

Contents

Tables

1. OVERVIEW

1.1. DOCUMENT STRUCTURE.

Document contains brief description of DP80390 instructions. This manual is intended for design engineers who are planning to use the DP80390 HDL core in conjunction with software assembler, compiler and debugger tools.

2. INSTRUCTIONS SET BRIEF

2.1. INSTRUCTION SET NOTES

The DP80390 has five different addressing modes: immediate, direct, register, indirect and relative. In the immediate addressing mode the data is contained in the opcode. By direct addressing an eight bit address is a part of the opcode, by register addressing, a register is selected in the opcode for the operation. In the indirect addressing mode, a register is selected in the opcode to point to the address used by the operation. The relative addressing mode is used for jump instructions.
The following tables give a survey about the instruction set cycles of the DP80390 microcontroller core. **One cycle is equal to one clock period**.
Table 1 and Table 2 contain notes for mnemonics used in Instruction set tables. Tables 3 - 7 show instruction hexadecimal codes, number of bytes and machine cycles that each instruction takes to execute.

Rn	Working register R0-R7
direct	128 internal RAM locations, any Special Function Registers
@Ri	Indirect internal or external RAM location addressed by register R0 or R1
#data	8-bit constant included in instruction
#data16	16-bit constant included as bytes 2 and 3 of instruction
#data24	24-bit constant included as bytes 2,3 and 4 of instruction
bit	256 software flags, any bit-addressable I/O pin, control or status bit
A	Accumulator

Table 1. Notes on data addressing modes

addr24	Destination address for LCALL and LJMP may be anywhere within the 16 MB of program memory address space in FLAT mode.
addr19	Destination address for ACALL and AJMP will be within the same 512 KB page of program memory as the first byte of the following instruction in FLAT mode
addr16	Destination address for LCALL and LJMP may be anywhere within the 64 kB of program memory address space in LARGE mode.
addr11	Destination address for ACALL and AJMP will be within the same 2 KB page of program memory as the first byte of the following instruction in LARGE mode
rel	SJMP and all conditional jumps include an 8-bit offset byte. Range is +127/-128 bytes relative to the first byte of the following instruction

Table 2. Notes on program addressing modes

2.2. INSTRUCTION SET BRIEF – FUNCTIONAL ORDER

2.2.1. ARITHMETIC OPERATIONS

Mnemonic	Description	Code	Bytes	Cycles
ADD A,Rn	Add register to accumulator	0x28-0x2F	1	1
ADD A,direct	Add direct byte to accumulator	0x25	2	2
ADD A,@Ri	Add indirect RAM to accumulator	0x26-0x27	1	2
ADD A,#data	Add immediate data to accumulator	0x24	2	2
ADDC A,Rn	Add register to accumulator with carry flag	0x38-0x3F	1	1
ADDC A,direct	Add direct byte to A with carry flag	0x35	2	2
ADDC A,@Ri	Add indirect RAM to A with carry flag	0x36-0x37	1	2
ADDC A,#data	Add immediate data to A with carry flag	0x34	2	2
SUBB A,Rn	Subtract register from A with borrow	0x98-0x9F	1	1
SUBB A,direct	Subtract direct byte from A with borrow	0x95	2	2
SUBB A,@Ri	Subtract indirect RAM from A with borrow	0x96-0x97	1	2
SUBB A,#data	Subtract immediate data from A with borrow	0x94	2	2
INC A	Increment accumulator	0x04	1	1
INC Rn	Increment register	0x08-0x0F	1	2
INC direct	Increment direct byte	0x05	2	3
INC @Ri	Increment indirect RAM	0x06-0x07	1	3
DEC A	Decrement accumulator	0x14	1	1
DEC Rn	Decrement register	0x18-0x1F	1	2
DEC direct	Decrement direct byte	0x15	1	3
DEC @Ri	Decrement indirect RAM	0x16-0x17	2	3
INC DPTR	Increment data pointer	0xA3	1	1
MUL A,B	**Multiply A and B**	**0xA4**	**1**	**2**
DIV A,B	**Divide A by B**	**0x84**	**1**	**6**
DA A	Decimal adjust accumulator	0xD4	1	3

Table 3. Arithmetic operations

2.2.2. Logic operations

Mnemonic	Description	Code	Bytes	Cycles
ANL A,Rn	AND register to accumulator	0x58-0x5F	1	1
ANL A,direct	AND direct byte to accumulator	0x55	2	2
ANL A,@Ri	AND indirect RAM to accumulator	0x56-0x57	1	2
ANL A,#data	AND immediate data to accumulator	0x54	2	2
ANL direct,A	AND accumulator to direct byte	0x52	2	3
ANL direct,#data	AND immediate data to direct byte	0x53	3	3
ORL A,Rn	OR register to accumulator	0x48-0x4F	1	1
ORL A,direct	OR direct byte to accumulator	0x45	2	2
ORL A,@Ri	OR indirect RAM to accumulator	0x46-0x47	1	2
ORL A,#data	OR immediate data to accumulator	0x44	2	2
ORL direct,A	OR accumulator to direct byte	0x42	2	3
ORL direct,#data	OR immediate data to direct byte	0x43	3	3
XRL A,Rn	Exclusive OR register to accumulator	0x68-0x6F	1	1
XRL A,direct	Exclusive OR direct byte to accumulator	0x65	2	2
XRL A,@Ri	Exclusive OR indirect RAM to accumulator	0x66-0x67	1	2
XRL A,#data	Exclusive OR immediate data to accumulator	0x64	2	2
XRL direct,A	Exclusive OR accumulator to direct byte	0x62	2	3
XRL direct,#data	Exclusive OR immediate data to direct byte	0x63	3	3
CLR A	Clear accumulator	0xE4	1	1
CPL A	Complement accumulator	0xF4	1	1
RL A	Rotate accumulator left	0x23	1	1
RLC A	Rotate accumulator left through carry	0x33	1	1
RR A	Rotate accumulator right	0x03	1	1
RRC A	Rotate accumulator right through carry	0x13	1	1
SWAP A	Swap nibbles within the accumulator	0xC4	1	1

Table 4. Logic operations

2.2.3. Boolean manipulation

Mnemonic	Description	Code	Bytes	Cycles
CLR C	Clear carry flag	0xC3	1	1
CLR bit	Clear direct bit	0xC2	2	3
SETB C	Set carry flag	0xD3	1	1
SETB bit	Set direct bit	0xD2	2	3
CPL C	Complement carry flag	0xB3	1	1
CPL bit	Complement direct bit	0xB2	2	3
ANL C,bit	AND direct bit to carry flag	0x82	2	2
ANL C,/bit	AND complement of direct bit to carry	0xB0	2	2
ORL C,bit	OR direct bit to carry flag	0x72	2	2
ORL C,/bit	OR complement of direct bit to carry	0xA0	2	2
MOV C,bit	Move direct bit to carry flag	0xA2	2	2
MOV bit,C	Move carry flag to direct bit	0x92	2	3

Table 5. Boolean manipulation

2.2.4. DATA TRANSFERS

<table>
<tr><th>Mnemonic</th><th colspan="3">Description</th><th>Code</th><th>Bytes</th><th>Cycles</th></tr>
<tr><td>MOV A,Rn</td><td colspan="3">Move register to accumulator</td><td>0xE8-0xEF</td><td>1</td><td>1</td></tr>
<tr><td>MOV A,direct</td><td colspan="3">Move direct byte to accumulator</td><td>0xE5</td><td>2</td><td>2</td></tr>
<tr><td>MOV A,@Ri</td><td colspan="3">Move indirect RAM to accumulator</td><td>0xE6-0xE7</td><td>1</td><td>2</td></tr>
<tr><td>MOV A,#data</td><td colspan="3">Move immediate data to accumulator</td><td>0x74</td><td>2</td><td>2</td></tr>
<tr><td>MOV Rn,A</td><td colspan="3">Move accumulator to register</td><td>0xF8-0xFF</td><td>1</td><td>1</td></tr>
<tr><td>MOV Rn,direct</td><td colspan="3">Move direct byte to register</td><td>0xA8-0xAF</td><td>2</td><td>3</td></tr>
<tr><td>MOV Rn,#data</td><td colspan="3">Move immediate data to register</td><td>0x78-0x7F</td><td>2</td><td>2</td></tr>
<tr><td>MOV direct,A</td><td colspan="3">Move accumulator to direct byte</td><td>0xF5</td><td>2</td><td>2</td></tr>
<tr><td>MOV direct,Rn</td><td colspan="3">Move register to direct byte</td><td>0x88-8F</td><td>2</td><td>2</td></tr>
<tr><td>MOV direct1,direct2</td><td colspan="3">Move direct byte to direct byte</td><td>85</td><td>3</td><td>3</td></tr>
<tr><td>MOV direct,@Ri</td><td colspan="3">Move indirect RAM to direct byte</td><td>86-87</td><td>2</td><td>3</td></tr>
<tr><td>MOV direct,#data</td><td colspan="3">Move immediate data to direct byte</td><td>75</td><td>3</td><td>3</td></tr>
<tr><td>MOV @Ri,A</td><td colspan="3">Move accumulator to indirect RAM</td><td>F6-F7</td><td>1</td><td>2</td></tr>
<tr><td>MOV @Ri,direct</td><td colspan="3">Move direct byte to indirect RAM</td><td>A6-A7</td><td>2</td><td>3</td></tr>
<tr><td>MOV @Ri,#data</td><td colspan="3">Move immediate data to indirect RAM</td><td>76-77</td><td>2</td><td>2</td></tr>
<tr><td>MOV DPTR,#data16</td><td colspan="3">Load 16-bit constant into active DPH and DPL in LARGE mode</td><td>90</td><td>3</td><td>3</td></tr>
<tr><td>MOV DPTR,#data24</td><td colspan="3">Load 24-bit constant into active DPX, DPH and DPL in FLAT mode</td><td>90</td><td>4</td><td>4</td></tr>
<tr><td>MOVC A,@A+DPTR</td><td colspan="3">Move code byte relative to DPTR to accumulator</td><td>93</td><td>1</td><td>5</td></tr>
<tr><td>MOVC A,@A+PC</td><td colspan="3">Move code byte relative to PC to accumulator</td><td>83</td><td>1</td><td>4</td></tr>
<tr><td>MOVX A,@Ri</td><td colspan="3">Move external RAM (8-bit address) to A</td><td>E2-E3</td><td>1</td><td>3*</td></tr>
<tr><td>MOVX A,@DPTR</td><td colspan="3">Move external RAM (16-bit address) to A</td><td>E0</td><td>1</td><td>2*</td></tr>
<tr><td rowspan="2">MOVX @Ri,A</td><td rowspan="2">Move A to external RAM (8-bit address)</td><td colspan="2">CODE inside ROM/RAM destination XRAM data</td><td rowspan="2">F2-F3</td><td rowspan="2">1</td><td>4*</td></tr>
<tr><td colspan="2">all other cases</td><td>5*</td></tr>
<tr><td rowspan="2">MOVX @DPTR,A</td><td rowspan="2">Move A to external RAM (16-bit address)</td><td colspan="2">CODE inside ROM/RAM destination XRAM data</td><td rowspan="2">F0</td><td rowspan="2">1</td><td>3*</td></tr>
<tr><td colspan="2">all other cases</td><td>4*</td></tr>
<tr><td rowspan="2">PUSH direct</td><td colspan="2" rowspan="2">Push direct byte onto stack</td><td>LARGE</td><td rowspan="2">C0</td><td rowspan="2">2</td><td>3</td></tr>
<tr><td>FLAT</td><td>3</td></tr>
<tr><td rowspan="2">POP direct</td><td colspan="2" rowspan="2">Pop direct byte from stack</td><td>LARGE</td><td rowspan="2">D0</td><td rowspan="2">2</td><td>2</td></tr>
<tr><td>FLAT</td><td>2</td></tr>
<tr><td>XCH A,Rn</td><td colspan="3">Exchange register with accumulator</td><td>C8-CF</td><td>1</td><td>2</td></tr>
<tr><td>XCH A,direct</td><td colspan="3">Exchange direct byte with accumulator</td><td>C5</td><td>2</td><td>3</td></tr>
<tr><td>XCH A,@Ri</td><td colspan="3">Exchange indirect RAM with accumulator</td><td>C6-C7</td><td>1</td><td>3</td></tr>
<tr><td>XCHD A,@Ri</td><td colspan="3">Exchange low-order nibble indirect RAM with A</td><td>D6-D7</td><td>1</td><td>3</td></tr>
</table>

Table 6. Data transfer

* MOVX cycles depends on external data memory access time (READY pin)

2.2.5. PROGRAM BRANCHES

Mnemonic	Description		Code	Bytes	Cycles
ACALL addr11	Absolute subroutine call	LARGE	0x11-0xF1	2	4
ACALL addr19		FLAT		3	5
LCALL addr16	Long subroutine call	LARGE	03	3	4
LCALL addr24		FLAT		4	6
RET	Return from subroutine	LARGE	22	1	4
		FLAT			5
RETI	Return from interrupt	LARGE	32	1	4
		FLAT			5
AJMP addr11	Absolute jump	LARGE	01-E1	2	3
AJMP addr19		FLAT		3	4
LJMP addr16	Long jump	LARGE	02	3	4
LJMP addr24		FLAT		4	5
SJMP rel	Short jump (relative address)		80	2	3
JMP @A+DPTR	Jump indirect relative to the DPTR		73	1	5
JZ rel	Jump if accumulator is zero		60	2	4
JNZ rel	Jump if accumulator is not zero		70	2	4
JC rel	Jump if carry flag is set		40	2	3
JNC rel	Jump if carry flag is not set		50	2	3
JB bit,rel	Jump if direct bit is set		20	3	5
JNB bit,rel	Jump if direct bit is not set		30	3	5
JBC bit,direct rel	Jump if direct bit is set and clear bit		10	3	5
CJNE A,direct rel	Compare direct byte to A and jump if not equal		B5	3	5
CJNE A,#data rel	Compare immediate to A and jump if not equal		B4	3	4
CJNE Rn,#data rel	Compare immediate to reg. and jump if not equal		B8-BF	3	4
CJNE @Ri,#data rel	Compare immediate to ind. and jump if not equal		B6-B7	3	5
DJNZ Rn,rel	Decrement register and jump if not zero		D8-DF	2	4
DJNZ direct,rel	Decrement direct byte and jump if not zero		D5	3	5
NOP	No operation		00	1	1

Table 7. Program branches

2.3. INSTRUCTION SET BRIEF – HEXADECIMAL ORDER

Opcode	Mnemonic	Opcode	Mnemonic
00 H	NOP	30 H	JNB bit.rel
01 H	AJMP addr11/addr19	31 H	ACALL addr11/addr19
02 H	LJMP addr16 /addr24	32 H	RETI
03 H	RR A	33 H	RLC A
04 H	INC A	34 H	ADDC A,#data
05 H	INC direct	35 H	ADDC A,direct
06 H	INC @R0	36 H	ADDC A,@R0
07 H	INC @R1	37 H	ADDC A,@R1
08 H	INC R0	38 H	ADDC A,R0
09 H	INC R1	39 H	ADDC A,R1
0A H	INC R2	3A H	ADDC A,R2
0B H	INC R3	3B H	ADDC A,R3
0C H	INC R4	3C H	ADDC A,R4
0D H	INC R5	3D H	ADDC A,R5
0E H	INC R6	3E H	ADDC A,R6
0F H	INC R7	3F H	ADDC A,R7
10 H	JBC bit,rel	40 H	JC rel
11 H	ACALL addr11/addr19	41 H	AJMP addr11/addr19
12 H	LCALL addr16/addr24	42 H	ORL direct,A
13 H	RRC A	43 H	ORL direct,#data
14 H	DEC A	44 H	ORL A,#data
15 H	DEC direct	45 H	ORL A,direct
16 H	DEC @R0	46 H	ORL A,@R0
17 H	DEC @R1	47 H	ORL A,@R1
18 H	DEC R0	48 H	ORL A,R0
19 H	DEC R1	49 H	ORL A,R1
1A H	DEC R2	4A H	ORL A,R2
1B H	DEC R3	4B H	ORL A,R3
1C H	DEC R4	4C H	ORL A,R4
1D H	DEC R5	4D H	ORL A,R5
1E H	DEC R6	4E H	ORL A,R6
1F H	DEC R7	4F H	ORL A,R7
20 H	JB bit.rel	50 H	JNC rel
21 H	AJMP addr11/addr19	51 H	ACALL addr11/addr19
22 H	RET	52 H	ANL direct,A
23 H	RL A	53 H	ANL direct,#data
24 H	ADD A,#data	54 H	ANL A,#data
25 H	ADD A,direct	55 H	ANL A,direct
26 H	ADD A,@R0	56 H	ANL A,@R0
27 H	ADD A,@R1	57 H	ANL A,@R1
28 H	ADD A,R0	58 H	ANL A,R0
29 H	ADD A,R1	59 H	ANL A,R1
2A H	ADD A,R2	5A H	ANL A,R2
2B H	ADD A,R3	5B H	ANL A,R3
2C H	ADD A,R4	5C H	ANL A,R4
2D H	ADD A,R5	5D H	ANL A,R5
2E H	ADD A,R6	5E H	ANL A,R6
2F H	ADD A,R7	5F H	ANL A,R7

Opcode	Mnemonic	Opcode	Mnemonic
60 H	JZ rel	90 H	MOV DPTR,#data16 MOV DPTR,#data24
61 H	AJMP addr11	91 H	ACALL addr11
62 H	XRL direct,A	92 H	MOV bit,C
63 H	XRL direct,#data	93 H	MOVC A,@A+DPTR
64 H	XRL A,#data	94 H	SUBB A,#data
65 H	XRL A,direct	95 H	SUBB A,direct
66 H	XRL A,@R0	96 H	SUBB A,@R0
67 H	XRL A,@R1	97 H	SUBB A,@R1
68 H	XRL A,R0	98 H	SUBB A,R0
69 H	XRL A,R1	99 H	SUBB A,R1
6A H	XRL A,R2	9A H	SUBB A,R2
6B H	XRL A,R3	9B H	SUBB A,R3
6C H	XRL A,R4	9C H	SUBB A,R4
6D H	XRL A,R5	9D H	SUBB A,R5
6E H	XRL A,R6	9E H	SUBB A,R6
6F H	XRL A,R7	9F H	SUBB A,R7
70 H	JNZ rel	A0 H	ORL C,bit
71 H	ACALL addr11/addr19	A1 H	AJMP addr11/addr19
72 H	ORL C,direct	A2 H	MOV C,bit
73 H	JMP @A+DPTR	A3 H	INC DPTR
74 H	MOV A,#data	A4 H	MUL AB
75 H	MOV direct,#data	A5 H	-
76 H	MOV @R0,#data	A6 H	MOV @R0,direct
77 H	MOV @R1,#data	A7 H	MOV @R1,direct
78 H	MOV R0.#data	A8 H	MOV R0,direct
79 H	MOV R1.#data	A9 H	MOV R1,direct
7A H	MOV R2.#data	AA H	MOV R2,direct
7B H	MOV R3.#data	AB H	MOV R3,direct
7C H	MOV R4.#data	AC H	MOV R4,direct
7D H	MOV R5.#data	AD H	MOV R5,direct
7E H	MOV R6.#data	AE H	MOV R6,direct
7F H	MOV R7.#data	AF H	MOV R7,direct
80 H	SJMP rel	B0 H	ANL C,bit
81 H	AJMP addr11/addr19	B1 H	ACALL addr11/addr19
82 H	ANL C,bit	B2 H	CPL bit
83 H	MOVC A,@A+PC	B3 H	CPL C
84 H	DIV AB	B4 H	CJNE A,#data,rel
85 H	MOV direct,direct	B5 H	CJNE A,direct,rel
86 H	MOV direct,@R0	B6 H	CJNE @R0,#data,rel
87 H	MOV direct,@R1	B7 H	CJNE @R1,#data,rel
88 H	MOV direct,R0	B8 H	CJNE R0,#data,rel
89 H	MOV direct,R1	B9 H	CJNE R1,#data,rel
8A H	MOV direct,R2	BA H	CJNE R2,#data,rel
8B H	MOV direct,R3	BB H	CJNE R3,#data,rel
8C H	MOV direct,R4	BC H	CJNE R4,#data,rel
8D H	MOV direct,R5	BD H	CJNE R5,#data,rel
8E H	MOV direct,R6	BE H	CJNE R6,#data,rel
8F H	MOV direct,R7	BF H	CJNE R7,#data,rel

Opcode	Mnemonic	Opcode	Mnemonic
C0 H	PUSH direct	E0 H	MOVX A,@DPTR
C1 H	AJMP addr11/addr19	E1 H	AJMP addr11/addr19
C2 H	CLR bit	E2 H	MOVX A,@R0
C3 H	CLR C	E3 H	MOVX A,@R1
C4 H	SWAP A	E4 H	CLR A
C5 H	XCH A, direct	E5 H	MOV A, direct
C6 H	XCH A,@R0	E6 H	MOV A,@R0
C7 H	XCH A,@R1	E7 H	MOV A,@R1
C8 H	XCH A,R0	E8 H	MOV A,R0
C9 H	XCH A,R1	E9 H	MOV A,R1
CA H	XCH A,R2	EA H	MOV A,R2
CB H	XCH A,R3	EB H	MOV A,R3
CC H	XCH A,R4	EC H	MOV A,R4
CD H	XCH A,R5	ED H	MOV A,R5
CE H	XCH A,R6	EE H	MOV A,R6
CF H	XCH A,R7	EF H	MOV A,R7
D0 H	POP direct	F0 H	MOVX @DPTR,A
D1 H	ACALL addr11/addr19	F1 H	ACALL addr11/addr19
D2 H	SETB bit	F2 H	MOVX @R0,A
D3 H	SETB C	F3 H	MOVX @R1,A
D4 H	DA A	F4 H	CPL A
D5 H	DJNZ direct, rel	F5 H	MOV direct, A
D6 H	XCHD A,@R0	F6 H	MOV @R0,A
D7 H	XCHD A,@R1	F7 H	MOV @R1,A
D8 H	DJNZ R0,rel	F8 H	MOV R0,A
D9 H	DJNZ R1,rel	F9 H	MOV R1,A
DA H	DJNZ R2,rel	FA H	MOV R2,A
DB H	DJNZ R3,rel	FB H	MOV R3,A
DC H	DJNZ R4,rel	FC H	MOV R4,A
DD H	DJNZ R5,rel	FD H	MOV R5,A
DE H	DJNZ R6,rel	FE H	MOV R6,A
DF H	DJNZ R7,rel	FF H	MOV R7,A

Table 8. Instruction set brief in hexadecimal order

3. INSTRUCTIONS SET DETAILS

3.1. ACALL *

3.1.1. LARGE

Instruction: 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, opcode bits 7-5, the second byte of the instruction. The subroutine called must therefore start within the same 2K block of program memory as the first byte of the instruction following ACALL. No flags are affected.

Operation:

(PC)	← (PC) + 2
(SP)	← (SP) + 1
((SP))	← (PC7-0)
(SP)	← (SP) + 1
((SP))	← (PC15-8)
(PC10-0)	← page address

Bytes: 2
Cycles: 4

Encoding:

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

3.1.2. FLAT

Instruction: ACALL addr19

Function: Absolute call

Description: ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC triple to obtain the address of the following instruction, then pushes the 24-bit result onto the stack (low-order byte first) and increments the stack pointer triple. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, opcode bits 7-5, the second and third byte of the instruction. The subroutine called must therefore start within the same 512K block of program memory as the first byte of the instruction following ACALL. No flags are affected.

Operation:

(PC) ← (PC) + 3
(SP) ← (SP) + 1
((SP)) ← (PC7-0)
(SP) ← (SP) + 1
((SP)) ← (PC15-8)
(SP) ← (SP) + 1
((SP)) ← (PC23-16)
(PC18-0) ← page address

Bytes: 3
Cycles: 5

Encoding:

a18	a17	a16	1	0	0	0	1
a15	a14	a13	a12	a11	a10	a9	a8
a7	a6	a5	a4	a3	a2	a1	a0

*** instruction modified regarding to standard 80C51**

3.2. ADD

Instruction: ADD A, <src-byte>

Function: Adds A to the source operand and returns the result to A.

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

3.2.1. ADD A, RN

Operation: (PC) ← (PC) + 1
(A) ← (A) + (Rn)

Bytes: 1
Cycles: 1

Encoding:

0	0	1	0	1	r	r	r

3.2.2. ADD A, DIRECT

Operation: (PC) ← (PC) + 2
(A) ← (A) + (direct)

Bytes: 2
Cycles: 2

Encoding:

0	0	1	0	0	1	0	1	direct address

3.2.3. ADD A, @Ri

Operation: (PC) ← (PC) + 1
(A) ← (A) + ((Ri))

Bytes: 1
Cycles: 2

Encoding:

0	0	1	0	0	1	1	i

3.2.4. ADD A, #data

Operation: (PC) ← (PC) + 2
(A) ← (A) + #data

Bytes: 2
Cycles: 2

Encoding:

0	0	1	0	0	1	0	0	immediate data

3.3. ADDC

Instruction: ADDC A, < src-byte>

Function: Adds A and the source operand, then adds one (1) if CY is set, and puts the result in A.

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

3.3.1. ADDC A, RN

Operation: (PC) ← (PC) + 1
(A) ← (A) + (C) + (Rn)

Bytes: 1
Cycles: 1

Encoding:

0	0	1	1	1	r	r	r

3.3.2. ADDC A, DIRECT

Operation: (PC) ← (PC) + 2
(A) ← (A) + (C) + (direct)

Bytes: 2
Cycles: 2

Encoding:

0	0	1	1	0	1	0	1	direct address

3.3.3. ADDC A, @Ri

Operation: (PC) ← (PC) + 1
(A) ← (A) + (C) + ((Ri))

Bytes: 1
Cycles: 2

Encoding:

0	0	1	1	0	1	1	i

3.3.4. ADDC A, #data

Operation: (PC) ← (PC) + 2
(A) ← (A) + (C) + #data

Bytes: 2
Cycles: 2

Encoding:

0	0	1	1	0	1	0	0	immediate data

3.4. AJMP *

3.4.1. LARGE

Instruction: AJMP addr11

Function: Absolute jump

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

Operation: (PC) ← (PC) + 2
(PC10-0) ← page address

Bytes: 2
Cycles: 3

Encoding:

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

3.4.2. FLAT

Instruction: AJMP addr19

Function: Absolute jump

Description: AJMP transfers program execution to the indicated address, which is formed at runtime by concatenating the high-order five bits of the PC (*after* incrementing the PC triple), opcode bits 7-5, the second and the third byte of the instruction. The destination must therefore be within the same 512K block of program memory as the first byte of the instruction following AJMP.

Operation: (PC) ← (PC) + 3
(PC18-0) ← page address

Bytes: 3
Cycles: 4

Encoding:

a18	a17	a16	0	0	0	0	1
a15	a14	a13	a12	a11	a10	a9	a8
a7	a6	a5	a4	a3	a2	a1	a0

*** instruction modified regarding to standard 80C51**

3.5. ANL

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

Function: Logical AND for byte operands

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 a 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.

3.5.1. ANL A, RN

Operation: (PC) ← (PC) + 1
(A) ← (A) and (Rn)

Bytes: 1
Cycles: 1

Encoding:

0	1	0	1	1	r	r	r

3.5.2. ANL A, DIRECT

Operation: (PC) ← (PC) + 2
(A) ← (A) and (direct)

Bytes: 2
Cycles: 2

Encoding:

0	1	0	1	0	1	0	1	direct address

3.5.3. ANL A, @Ri

Operation: (PC) ← (PC) + 1
(A) ← (A) and ((Ri))

Bytes: 1
Cycles: 2

Encoding:

0	1	0	1	0	1	1	i

3.5.4. ANL A, #data

Operation: (PC) ← (PC) + 2
(A) ← (A) and #data

Bytes: 2
Cycles: 2

Encoding:

0	1	0	1	0	1	0	0	immediate data

3.5.5. ANL direct, A

Operation: (PC) ← (PC) + 2
(direct) ← (direct) and (A)

Bytes: 2
Cycles: 3

Encoding:

0	1	0	1	0	0	1	0	direct address

3.5.6. ANL direct, #data

Operation: (PC) ← (PC) + 3
(direct) ← (direct) and #data

Bytes: 3
Cycles: 3

Encoding:

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

Instruction: ANL C, <src-bit>

Function: Logical AND for bit operands

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

3.5.7. ANL C, BIT

Operation: (PC) ← (PC) + 2
(C) ← (C) and (bit)

Bytes: 2
Cycles: 2

Encoding:

1	0	0	0	0	0	1	0	bit address

3.5.8. ANL C, /BIT

Operation: (PC) ← (PC) + 2
(C) ← (C) and / (bit)

Bytes: 2
Cycles: 2

Encoding:

1	0	1	1	0	0	0	0	bit address

3.6. CJNE

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

Function: Compare and jump if not equal.

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

3.6.1. 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
Cycles: 5

Encoding:

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

3.6.2. 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
Cycles: 4

Encoding:

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

3.6.3. 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
Cycles: 4

Encoding:

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

3.6.4. 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
Cycles: 5

Encoding:

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

3.7. CLR

3.7.1. CLR A

Function: Clear accumulator

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

Operation: (PC) ← (PC) + 1
(A) ← 0

Bytes: 1
Cycles: 1

Encoding:

1	1	1	0	0	1	0	0

3.7.2. CLR BIT

Function: Clear bit

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

Operation: (PC) ← (PC) + 2
bit ← 0

Bytes: 2
Cycles: 3

Encoding:

1	1	0	0	0	0	1	0	bit address

3.7.3. CLR C

Function: Clear carry

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

Operation: (PC) $\leftarrow$ (PC) + 1
(C) $\leftarrow$ 0

Bytes: 1
Cycles: 1

Encoding:

1	1	0	0	0	0	1	1

3.8. CPL

3.8.1. CPL A

Function: Complement accumulator

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

Operation: (PC) ← (PC) + 1
(A) ← / (A)

Bytes: 1
Cycles: 1

Encoding:

1	1	1	1	0	1	0	0

3.8.2. CPL BIT

Function: Complement bit

Description: The bit variable specified is complemented. A bit which had been a one is changed to zero and vice versa. No other flags are affected. CPL can operate on 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 will be read from the output data latch, not the input pin.

Operation: (PC) ← (PC) + 2
(C) ← (bit)

Bytes: 2
Cycles: 3

Encoding:

1	0	1	1	0	0	1	0	bit address

3.8.3. CPL C

Function: Complement carry

Description: The carry flag is complemented. A bit which had been a one is changed to zero and vice versa.

Operation: (PC) $\leftarrow$ (PC) + 1
(C) $\leftarrow$ / (C)

Bytes: 1
Cycles: 1

Encoding:

1	0	1	1	0	0	1	1

3.9. DA

Instruction: DA A

Function: Decimal adjust accumulator for addition

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

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

All of this occurs during the one instruction cycle. Essentially; this instruction performs the decimal conversion by adding 00 H , 06 H , 60 H , or 66 H 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:

(PC) ← (PC) + 1
if [[(A3-0) > 9] ^ [(AC) = 1]] then
  (A3-0) ← (A3-0) + 6
next
if [[(A7-4) > 9] ^ [(C) = 1]] then
  (A7-4) ← (A7-4) + 6

Bytes: 1
Cycles: 3

Encoding:

1	1	0	1	0	1	0	0

3.10. DEC

Instruction: DEC byte

Function: Decrement byte

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

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

3.10.1. DEC A

Operation: (PC) ← (PC) + 1
(A) ← (A) - 1

Bytes: 1
Cycles: 1

Encoding:

0	0	0	1	0	1	0	0

3.10.2. DEC RN

Operation: (PC) ← (PC) + 1
(Rn) ← (Rn) - 1

Bytes: 1
Cycles: 2

Encoding:

0	0	0	1	1	r	r	r

3.10.3. DEC DIRECT

Operation: (PC) ← (PC) + 2
(direct) ← (direct) - 1

Bytes: 2
Cycles: 3

Encoding:

0	0	0	1	0	1	0	1	direct address

3.10.4. DEC @RI

Operation: (PC) ← (PC) + 1
((Ri)) ← ((Ri)) - 1

Bytes: 1
Cycles: 3

Encoding:

0	0	0	1	0	1	1	i

3.11. DIV

Instruction: DIV AB

Function: Divide

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

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

Operation: (PC) ← (PC) + 1
(A15-8) ← (A) / (B) – result's bits 15..8
(B7-0) ← (A) / (B) – result's bits 7..0

Bytes: 1
Cycles: 6

Encoding:

1	0	0	0	0	1	0	0

3.12. DJNZ

Instruction: 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 00 H will underflow to 0FF H. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction. The location decremented may be a register or directly addressed byte.

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

3.12.1. DJNZ RN, REL

Operation:
(PC) $\leftarrow$ (PC) + 2
(Rn) $\leftarrow$ (Rn) - 1
if (Rn) $\neq$ 0 then
  (PC) $\leftarrow$ (PC) + rel

Bytes: 2
Cycles: 4

Encoding:

1	1	0	1	1	r	r	r	relative address

3.12.2. DJNZ DIRECT, REL

Operation: (PC) $\leftarrow$ (PC) + 3
(direct) $\leftarrow$ (direct) - 1
if (direct) $\neq$ 0 then
(PC) $\leftarrow$ (PC) + rel

Bytes: 3
Cycles: 5

Encoding:

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

3.13. INC

Instruction: INC operand

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 will be read from the output data latch, *not* the input pins.

3.13.1. INC A

Operation: (PC) ← (PC) + 1
(A) ← (A) + 1

Bytes: 1
Cycles: 1

Encoding:

0	0	0	0	0	1	0	0

3.13.2. INC RN

Operation: (PC) ← (PC) + 1
(Rn) ← (Rn) + 1

Bytes: 1
Cycles: 2

Encoding:

0	0	0	0	1	r	r	r

3.13.3. INC DIRECT

Operation: (PC) ← (PC) + 2
(direct) ← (direct) + 1

Bytes: 2
Cycles: 3

Encoding:

0	0	0	0	0	1	0	1	direct address

3.13.4. INC @RI

Operation: (PC) ← (PC) + 1
((Ri)) ← ((Ri)) + 1

Bytes: 1
Cycles: 3

Encoding:

0	0	0	0	0	1	1	i

3.13.5. INC DPTR*

Function: Increment active data pointer

Description: Increment the 16-bit data pointer (in LARGE) or 24-bit data pointer (in FLAT) by 1. A 16-bit/24-bit increment (modulo $2^{16}/2^{24}$) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0xFF to 0x00 will increment the high-order byte (DPH). No flags are affected. This is the only 16-bit/24-bit register which can be incremented or decremented. Refer to Data Pointer Extended registers chapter of DP80390 specification.

Operation: (PC) ← (PC) + 1
(DPTR) ← (DPTR) + 1

Bytes: 1
Cycles: 1

Encoding:

1	0	1	0	0	0	1	1

3.14. JB

Instruction: JB bit, rel

Function: Jump if bit is set

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

Operation: (PC) ← (PC) + 3
if (bit) = 1 then
  (PC) ← (PC) + rel

Bytes: 3
Cycles: 5

Encoding:

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

3.15. JBC

Instruction: JBC bit, rel

Function: Jump if bit is set and clear bit

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

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

Operation:
(PC) ← (PC) + 3
if (bit) = 1 then
  (bit) ← 0
  (PC) ← (PC) + rel

Bytes: 3
Cycles: 5

Encoding:

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

3.16. JC

Instruction: JC rel

Function: Jump if carry is set

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

Operation: (PC) $\leftarrow$ (PC) + 2
if (C) = 1 then
(PC) $\leftarrow$ (PC) + rel

Bytes: 2
Cycles: 3

Encoding:

0	1	0	0	0	0	0	0	relative address

3.17. JMP*

Instruction: JMP @A + DPTR

Function: **Jump indirect**

Description: Add the eight-bit unsigned contents of the accumulator with the 16-bit (in LARGE)/24-bit (in FLAT) active data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. 16-bit/24-bit addition is performed: 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: (PC) $\leftarrow$ (A) + (DPTR)

Bytes: 1
Cycles: 5

Encoding:

0	1	1	1	0	0	1	1

3.18. JNB

Instruction: JNB bit,rel

Function: Jump if bit is not set

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

Operation: (PC) ← (PC) + 3
if (bit) = 0 then
(PC) ← (PC) + rel.

Bytes: 3
Cycles: 5

Encoding:

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

3.19. JNC

Instruction: JNC rel

Function: Jump if carry is not set

Description: If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.

Operation: (PC) ← (PC) + 2
if (C) = 0 then
(PC) ← (PC) + rel

Bytes: 2
Cycles: 3

Encoding:

0	1	0	1	0	0	0	0	relative address

3.20. JNZ

Instruction: JNZ rel

Function: Jump if accumulator is not zero

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

Operation: $(PC) \leftarrow (PC) + 2$
if $(A) \neq 0$
then $(PC) \leftarrow (PC) + rel.$

Bytes: 2
Cycles: 4

Encoding:

0	1	1	1	0	0	0	0	relative address

3.21. JZ

Instruction: JZ rel

Function: Jump if accumulator is zero

Description: If all bits of the accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.

Operation: (PC) ← (PC) + 2
if (A) = 0 then
  (PC) ← (PC) + rel

Bytes: 2
Cycles: 4

Encoding:

0	1	1	0	0	0	0	0	relative address

3.22. LCALL *

3.22.1. LARGE

Instruction: LCALL addr16

Function: Long call

Description: LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the stack pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64 kB program memory address space. No flags are affected.

Operation:
(PC) ← (PC) + 3
(SP) ← (SP) + 1
((SP)) ← (PC7-0)
(SP) ← (SP) + 1
((SP)) ← (PC15-8)
(PC) ← addr15-0

Bytes: 3
Cycles: 4

Encoding:

0	0	0	1	0	0	1	0
address 15..8							
address 7..0							

3.22.2. FLAT

Instruction: LCALL addr24

Function: Long call

Description: LCALL calls a subroutine located at the indicated address. The instruction adds four to the program counter to generate the address of the next instruction and then pushes the 24-bit result onto the stack (low byte first), incrementing the stack pointer by three. The high-order and low-order bytes of the PC are then loaded, respectively, with the second, third and fourth bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 16 MB program memory address space. No flags are affected.

Operation:
(PC) ← (PC) + 4
(SP) ← (SP) + 1
((SP)) ← (PC7-0)
(SP) ← (SP) + 1
((SP)) ← (PC15-8)
(SP) ← (SP) + 1
((SP)) ← (PC23-16)
(PC) ← addr23-0

Bytes: 4
Cycles: 6

Encoding:

0	0	0	1	0	0	1	0
address 23..16							
address 15..8							
address 7..0							

*** instruction modified regarding to standard 80C51**

3.23. LJMP *

3.23.1. LARGE

Instruction: LJMP addr16

Function: Long jump

Description: LJMP causes an unconditional branch to the indicated address, by loading the PC with the second and third instruction bytes. The destination may therefore be anywhere in the full 64 kB program memory address space. No flags are affected.

Operation: (PC) $\leftarrow$ addr15... addr0

Bytes: 3
Cycles: 4

Encoding:

0	0	0	0	0	0	1	0
address 15..8							
address 7..0							

3.23.2. FLAT

Instruction: LCALL addr24

Function: Long jump

Description: LJMP causes an unconditional branch to the indicated address, by loading the PC with the second, third and fourth instruction bytes. The destination may therefore be anywhere in the full 16MB program memory address space. No flags are affected.

Operation: (PC) $\leftarrow$ addr23... addr0

Bytes: 4
Cycles: 6

Encoding:

0	0	0	0	0	0	1	0
address 23..16							
address 15..8							
address 7..0							

*** instruction modified regarding to standard 80C51**

3.24. MOV

Instruction: 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.

3.24.1. MOV A, Rn

Operation: (PC) ← (PC) + 1
(A) ← (Rn)

Bytes: 1
Cycles: 1

Encoding:

1	1	1	0	1	r	r	r

3.24.2. MOV A, direct

Operation: (PC) ← (PC) + 2
(A) ← (direct)

Note: MOV A, ACC is a **valid** instruction.

Bytes: 2
Cycles: 2

Encoding:

1	1	1	0	0	1	0	1	direct address

3.24.3. MOV A, @Ri

Operation: (PC) ← (PC) + 1

(A) ← ((Ri))

Bytes: 1
Cycles: 2

Encoding:

1	1	1	0	0	1	1	i

3.24.4. MOV A, #DATA

Operation: (PC) ← (PC) + 2
(A) ← #data

Bytes: 2
Cycles: 2

Encoding:

0	1	1	1	0	1	0	0	immediate data

3.24.5. MOV RN, A

Operation: (PC) ← (PC) + 1
(Rn) ← (A)

Bytes: 1
Cycles: 1

Encoding:

1	1	1	1	1	r	r	r

3.24.6. MOV RN, DIRECT

Operation: (PC) ← (PC) + 2
(Rn) ← (direct)

Bytes: 2
Cycles: 3

Encoding:

1	0	1	0	1	r	r	r	direct address

3.24.7. MOV RN, #DATA

Operation: (PC) ← (PC) + 2
(Rn) ← #data

Bytes: 2
Cycles: 2

Encoding:

0	1	1	1	1	r	r	r	immediate data

3.24.8. MOV DIRECT, A

Operation: (PC) ← (PC) + 2
(direct) ← (A)

Bytes: 2
Cycles: 2

Encoding:

1	1	1	1	0	1	0	1	direct address

3.24.9. MOV DIRECT, RN

Operation: (PC) ← (PC) + 2
(direct) ← (Rn)

Bytes: 2
Cycles: 2

Encoding:

1	0	0	0	1	r	r	r	direct address

3.24.10. MOV DIRECT, DIRECT

Operation: (PC) ← (PC) + 3
(direct) ← (direct)

Bytes: 3
Cycles: 3

Encoding:

1	0	0	0	0	1	0	1
direct address (source)							
direct address (destination)							

3.24.11. MOV DIRECT, @RI

Operation: (PC) ← (PC) + 2
(direct) ← ((Ri))

Bytes: 2
Cycles: 3

Encoding:

1	0	0	0	0	1	1	i	direct address

3.24.12. MOV DIRECT, #DATA

Operation: (PC) ← (PC) + 2
(direct) ← #data

Bytes: 3
Cycles: 3

Encoding:

0	1	1	1	0	1	0	1
direct address (source)							
immediate data							

3.24.13. MOV @RI, A

Operation: (PC) ← (PC) + 1
((Ri)) ← (A)

Bytes: 1
Cycles: 2

Encoding:

1	1	1	1	0	1	1	i

3.24.14. MOV @RI, DIRECT

Operation: (PC) ← (PC) + 2
((Ri)) ← (direct)

Bytes: 2
Cycles: 3

Encoding:

1	0	1	0	0	1	1	i	direct address

3.24.15. MOV @RI, #DATA

Operation: (PC) ← (PC) + 2
((Ri)) ← #data

Bytes: 2

Cycles: 2

Encoding:

0	1	1	1	0	1	1	i	immediate data

3.24.16. MOV C, BIT

Function: Move bit data

Description: The Boolean variable indicated by the second operand (directly addressable bit) is copied into carry flag. No other register or flag is affected.

Operation: (PC) ← (PC) + 2
(C) ← (bit)

Bytes: 2
Cycles: 2

Encoding:

1	0	1	0	0	0	1	0	bit address

3.24.17. MOV BIT, C

Function: Move carry flag

Description: The carry flag is copied into the Boolean variable indicated by the first operand (directly addressable bit). No other register or flag is affected.

Operation: (PC) ← (PC) + 2
(bit) ← (C)

Bytes: 2
Cycles: 3

Encoding:

1	0	0	1	0	0	1	0	bit address

3.24.18. MOV DPTR, #DATA16 - LARGE

Function: Load active data pointer with a 16-bit constant in LARGE mode

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

Operation: (PC) ← (PC) + 3
DPH ← immediate data15...8
DPL ← immediate data7..0

Bytes: 3
Cycles: 3

Encoding:

1	0	0	0	0	1	0	1
immediate data 15...8							
immediate data 7...0							

3.24.19. MOV DPTR, #DATA24* - FLAT

Function: Load active data pointer with a 24-bit constant in FLAT mode

Description: The active data pointer is loaded with the 24-bit constant indicated. The 24 bit constant is loaded into the second, third and fourth bytes of the instruction. The second byte (DPX or DPX1) is the high-order byte, while the third byte (DPH) holds the mid-order byte and fourth (DPL). No flags are affected. This is the only instruction which moves 24 bits of data at once.

Operation: (PC) ← (PC) + 4
DPX/DPX1 ← immediate data23...16
DPH ← immediate data15...8
DPL ← immediate data7..0

Bytes: 4
Cycles: 4

Encoding:

1	0	0	0	0	1	0	1
immediate data 23...16							
immediate data 15...8							
immediate data 7...0							

*** instruction modified regarding to standard 80C51**

3.25. MOVC*

Instruction: 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/24-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/24-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

3.25.1. MOVC A, @A + DPTR

Operation: (PC) ← (PC) + 1
(A) ← ((A) + (DPTR))

Bytes: 1
Cycles: 5

Encoding:

1	0	0	1	0	0	1	1

3.25.2. MOVC A, @A + PC

Operation: (PC) ← (PC) + 1
(A) ← ((A) + (PC))

Bytes: 1
Cycles: 4

Encoding:

1	0	0	0	0	0	1	1

*** different registers are used in FLAT and LARGE mode**

3.26. MOVX*

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

Function: Move external

Description: The MOVX instructions transfer data between the accumulator and a byte of external data memory, hence the X appended to MOV. There are two types of instructions, differing in whether they provide an 8-bit in both modes or 16-bit in LARGE/24-bit in FLAT indirect address to the external data RAM.
In the first type, the contents of R0 or R1 in the current register bank provides an eight-bit address, in the second type of MOVX instructions, the active data pointer generates 16-bit/24-bit address.

3.26.1. MOVX A, @Ri

Operation: (PC) ← (PC) + 1
(A) ← ((Ri))

Bytes: 1
Cycles: 3*

Encoding:

1	1	1	0	0	0	1	i

3.26.2. MOVX A, @DPTR

Operation: (PC) ← (PC) + 1
(A) ← ((DPTR))

Bytes: 1
Cycles: 2*

Encoding:

1	1	1	0	0	0	0	0

3.26.3. MOVX @Ri, A

Operation: (PC) ← (PC) + 1
((Ri)) ← (A)

Bytes: 1
Cycles: 4* – if MOVX CODE is executed from on-chip ROM or on-chip RAM, and destination data are placed inside off-chip XRAM
5* – for all other cases as follow:
CODE inside off-chip XPRG, destination inside off-chip XRAM
CODE inside off-chip XPRG, destination inside off-chip XPRG
CODE inside off-chip XPRG, destination inside on-chip PRG RAM
CODE inside on-chip ROM, destination inside off-chip XPRG
CODE inside on-chip ROM, destination inside on-chip PRG RAM
CODE inside on-chip RAM, destination inside off-chip XPRG
CODE inside on-chip RAM, destination inside on-chip PRG RAM

Encoding:

1	1	1	1	0	0	1	i

3.26.4. MOVX @DPTR, A

Operation: (PC) ← (PC) + 1
((DPTR)) ← (A)

Bytes: 1
Cycles: 3* – if MOVX CODE is executed from on-chip ROM or on-chip RAM, and destination data are placed inside off-chip XRAM
4* – for all other cases as listed above

Encoding:

1	1	1	1	0	0	0	0

* MOVX cycles depends on READY pin. Shown values with 0 Wait-States.

3.27. MUL

Instruction: 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 sixteen-bit product is left in the accumulator, and the high-order byte in B. If the product is greater than 255 (0FF H) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

Operation: (PC) ← (PC) + 1
(A) ← (A) x (B) – result's bits 7..0
(B) ← (A) x (B) – result's bits 15..8

Bytes: 1
Cycles: 2

Encoding:

1	0	1	0	0	1	0	0

3.28. NOP

Function: No operation

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

Operation: (PC) $\leftarrow$ (PC) + 1

Bytes: 1
Cycles: 1

Encoding:

0	0	0	0	0	0	0	0

3.29. ORL

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

Function: **Logical OR for byte variables**

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

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

3.29.1. ORL A, RN

Operation: (PC) ← (PC) + 1
(A) ← (A) or (Rn)

Bytes: 1
Cycles: 1

Encoding:

0	1	0	0	1	r	r	r

3.29.2. ORL A, DIRECT

Operation: (PC) ← (PC) + 2
(A) ← (A) or (direct)

Bytes: 2
Cycles: 2

Encoding:

0	1	0	0	0	1	0	1	direct address

3.29.3. ORL A, @Ri

Operation: (PC) ← (PC) + 1
(A) ← (A) or ((Ri))

Bytes: 1
Cycles: 2

Encoding:

0	1	0	0	0	1	1	i

3.29.4. ORL A, #DATA

Operation: (PC) ← (PC) + 1
(A) ← (A) or #data

Bytes: 2
Cycles: 2

Encoding:

0	1	0	0	0	1	0	0	immediate data

3.29.5. ORL DIRECT, A

Operation: (PC) ← (PC) + 1
(direct) ← (direct) or (A)

Bytes: 2
Cycles: 3

Encoding:

0	1	0	0	0	0	1	0	direct address

3.29.6. ORL DIRECT, #DATA

Operation: (PC) ← (PC) + 1
(direct) ← (direct) or #data

Bytes: 3
Cycles: 3

Encoding:

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

Instruction: ORL C, <src-bit>

Function: Logical OR for bit variables

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

3.29.7. ORL C, BIT

Operation: (PC) ← (PC) + 2
(C) ← (C) or (bit)

Bytes: 2
Cycles: 2

Encoding:

0	1	1	1	0	0	1	0	bit address

3.29.8. ORL C, /BIT

Operation: (PC) ← (PC) + 2
(C) ← (C) or /(bit)

Bytes: 2
Cycles: 2

Encoding:

1	0	1	0	0	0	0	0	bit address

3.30. POP*

3.30.1. LARGE

Instruction: 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 the transfer to the directly addressed byte indicated. No flags are affected.

Operation:
(PC) ← (PC) + 2
(direct) ← ((SP))
(SP) ← (SP) - 1

Bytes: 2
Cycles: 2

Encoding:

1	1	0	1	0	0	0	0	direct address

3.30.2. FLAT

Instruction: 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 the transfer to the directly addressed byte indicated. No flags are affected.

Operation:
(PC) ← (PC) + 2
(direct) ← ((SP))
(SP) ← (SP) - 1

Bytes: 2
Cycles: 2

Encoding:

1	1	0	1	0	0	0	0	direct address

*** instruction timing is identical in FLAT and LARGE mode**

3.31. PUSH*

3.31.1. LARGE

Instruction: 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:
(PC) ← (PC) + 2
(SP) ← (SP) + 1
((SP)) ← (direct)

Bytes: 2
Cycles: 3

Encoding:

1	1	0	0	0	0	0	0	direct address

3.31.2. FLAT

Instruction: 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:
(PC) ← (PC) + 2
(SP) ← (SP) + 1
((SP)) ← (direct)

Bytes: 2
Cycles: 3

Encoding:

1	1	0	0	0	0	0	0	direct address

*** instruction timing is identical in FLAT and LARGE mode**

3.32. RET *

3.32.1. LARGE

Function: Return from subroutine

Description: RET pops the PC successively from the stack, decrementing the stack pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.

Operation:
(PC15-8) ← ((SP))
(SP) ← (SP) - 1
(PC7-0) ← ((SP))
(SP) ← (SP) - 1

Bytes: 1
Cycles: 4

Encoding:

0	0	1	0	0	0	1	0

3.32.2. FLAT

Function: Return from subroutine

Description: RET pops the PC successively from the stack, decrementing the stack pointer by three. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.

Operation:
(PC23-16) ← ((SP))
(SP) ← (SP) - 1
(PC15-8) ← ((SP))
(SP) ← (SP) - 1
(PC7-0) ← ((SP))
(SP) ← (SP) - 1

Bytes: 1
Cycles: 5

Encoding:

0	0	1	0	0	0	1	0

*** instruction modified regarding to standard 80C51**

3.33. RETI *

3.33.1. LARGE

Function: Return from interrupt

Description: RETI pops 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 will be executed before the pending interrupt is processed.

Operation:

$(PC15\text{-}8) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$
$(PC7\text{-}0) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$

Bytes: 1
Cycles: 4

Encoding:

0	0	1	1	0	0	1	0

3.33.2. FLAT

Function: Return from interrupt

Description: RETI pops 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 three. 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 will be executed before the pending interrupt is processed.

Operation:
(PC23-16) ← ((SP))
(SP) ← (SP) - 1
(PC15-8) ← ((SP))
(SP) ← (SP) - 1
(PC7-0) ← ((SP))
(SP) ← (SP) - 1

Bytes: 1
Cycles: 5

Encoding:

0	0	1	1	0	0	1	0

*** instruction modified regarding to standard 80C51**

3.34. RL

Instruction: 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:
(PC) ← (PC) + 1
(An + 1) ← (An) n = 0-6
(A0) ← (A7)

Bytes: 1
Cycles: 1

Encoding:

0	0	1	0	0	0	1	1

3.35. RLC

Instruction: RLC A

Function: Rotate accumulator left through carry flag

Description: The eight bits in the accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.

Operation:
(PC) ← (PC) + 1
(An + 1) ← (An) n = 0-6
(A0) ← (C)
(C) ← (A7)

Bytes: 1
Cycles: 1

Encoding:

0	0	1	1	0	0	1	1

3.36. RR

Instruction: 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:
$(PC) \leftarrow (PC) + 1$
$(An) \leftarrow (An + 1)$ n = 0-6
$(A7) \leftarrow (A0)$

Bytes: 1
Cycles: 1

Encoding:

0	0	0	0	0	0	1	1

3.37. RRC

Instruction: RRC A

Function: Rotate accumulator right through carry flag

Description: The eight bits in the accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.

Operation:
(PC) ← (PC) + 1
(An) ← (An + 1) n=0-6
(A7) ← (C)
(C) ← (A0)

Bytes: 1
Cycles: 1

Encoding:

0	0	0	1	0	0	1	1

3.38. SETB

Instruction: SETB <bit>

Function: Set bit

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

3.38.1. SETB C

Operation: (PC) ← (PC) + 1
(C) ← 1

Bytes: 1
Cycles: 1

Encoding:

1	1	0	1	0	0	1	1

3.38.2. SETB BIT

Operation: (PC) ← (PC) + 2
(bit) ← 1

Bytes: 2
Cycles: 3

Encoding:

1	1	0	1	0	0	1	0	bit address

3.39. SJMP

Instruction: 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 102 H. Therefore, the displacement byte of the instruction will be the relative offset (0123 H - 0102 H) = 21 H . In other words, an SJMP with a displacement of 0FE H would be a one-instruction infinite loop.

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

Bytes: 2
Cycles: 3

Encoding:

1	0	0	0	0	0	0	0	relative address

3.40. SUBB

Instruction: SUBB A, <src-byte>

Function: Subtract with borrow

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

3.40.1. SUBB A, RN

Operation: (PC) ← (PC) + 1
(A) ← (A) - (C) - (Rn)

Bytes: 1
Cycles: 1

Encoding:

1	0	0	1	1	r	r	r

3.40.2. SUBB A, DIRECT

Operation: (PC) ← (PC) + 2
(A) ← (A) - (C) - (direct)

Bytes: 2
Cycles: 2

Encoding:

1	0	0	1	0	1	0	1	direct address

3.40.3. SUBB A, @Ri

Operation: (PC) ← (PC) + 1
(A) ← (A) - (C) - ((Ri))

Bytes: 1
Cycles: 2

Encoding:

1	0	0	1	0	1	1	i

3.40.4. SUBB A, #DATA

Operation: (PC) ← (PC) + 2
(A) ← (A) - (C) - #data

Bytes: 2
Cycles: 2

Encoding:

1	0	0	1	0	1	0	0	immediate data

3.41. SWAP

Instruction: SWAP A

Function: Swap nibbles within the accumulator

Description: SWAP A interchanges the low and high-order nibbles (four-bit fields) of the accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction. No flags are affected.

Operation: (PC) $\leftarrow$ (PC) + 1
(A3-0) $\leftrightarrow$ (A7-4),
(A7-4) $\leftrightarrow$ (A3-0)

Bytes: 1
Cycles: 1

Encoding:

1	1	0	0	0	1	0	0

3.42. XCH

Instruction: XCH A, <byte>

Function: Exchange accumulator with byte variable

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

3.42.1. XCH A, RN

Operation: (PC) ← (PC) + 1
(A) ↔ (Rn)

Bytes: 1
Cycles: 2

Encoding:

1	1	0	0	1	r	r	r

3.42.2. XCH A, DIRECT

Operation: (PC) ← (PC) + 2
(A) ↔ (direct)

Bytes: 2
Cycles: 3

Encoding:

1	1	0	0	0	1	0	1	direct address

3.42.3. XCH A, @RI

Operation: (PC) ← (PC) + 1
(A) ↔ ((Ri))

Bytes: 1
Cycles: 3

Encoding:

1	1	0	0	0	1	1	i

3.43. XCHD

Instruction: XCHD A, @Ri

Function: Exchange digit

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

Operation: (PC) $\leftarrow$ (PC) + 1
(A3-0) $\leftrightarrow$ ((Ri)3-0)

Bytes: 1
Cycles: 3

Encoding:

1	1	0	1	0	1	1	i

3.44. XRL

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

Function: Logical Exclusive OR for byte variables

Description: XRL performs the bit wise logical Exclusive OR operation between the indicated variables, storing the results in the destination. No flags are affected (except P, 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 will be read from the output data latch, *not* the input pins.

3.44.1. XRL A, RN

Operation: (PC) ← (PC) + 1
(A) ← (A) xor (Rn)

Bytes: 1
Cycles: 1

Encoding:

0	1	1	0	1	r	r	r

3.44.2. XRL A, DIRECT

Operation: (PC) ← (PC) + 2
(A) ← (A) xor (direct)

Bytes: 2
Cycles: 2

Encoding:

0	1	1	0	0	1	0	1	direct address

3.44.3. XRL A, @ Ri

Operation: (PC) ← (PC) + 1
(A) ← (A) xor ((Ri))

Bytes: 1
Cycles: 2

Encoding:

0	1	1	0	0	1	1	i

3.44.4. XRL A, #DATA

Operation: (PC) ← (PC) + 2
(A) ← (A) xor #data

Bytes: 2
Cycles: 2

Encoding:

0	1	1	0	0	1	0	0	immediate data

3.44.5. XRL DIRECT, A

Operation: (PC) ← (PC) + 2
(direct) ← (direct) xor (A)

Bytes: 2
Cycles: 3

Encoding:

0	1	1	0	0	0	1	0	direct address

3.44.6. XRL DIRECT, #DATA

Operation: (PC) ← (PC) + 3
(direct) ← (direct) xor #data

Bytes: 3
Cycles: 3

Encoding:

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

4. CONTACTS

If any problems are encountered please contact Digital Core Design.

Headquarters:

Wroclawska 94

41-902 Bytom

POLAND

e-mail: info@dcd.pl

tel. *:* +48 32 282 82 66

fax *:* +48 32 282 74 37

Field Office:

Texas Research Park

14815 Omicron Dr. suite 100

San Antonio, TX 78245, USA

e-mail: infoUS@dcd.pl

tel. *:* +1 210 422 8268

fax *:* +1 210 679 7511

Distributors:

Please check http://www.dcd.pl/apartn.php