Arithmetic Operations

Function	Instruction	OpCode/Byte/Cycle										Operation	PSW		
		b7	b6	b5	b4	b3	b2	b1	b0	#	~		CY	AC	OV
Add to Accumulator	ADD A,#data	0	0	1	0	0	1	0	0	2	1	A ← A + #data	×	×	×
	ADD A,direct	0	0	1	0	0	1	0	1	2	1	A ← A + (direct)	×	×	×
	ADD A,@Ri	0	0	1	0	0	1	1	**i**	1	1	A ← A + (Ri)	×	×	×
	ADD A,Rn	0	0	1	0	1	**r**	**r**	**r**	1	1	A ← A + Rn	×	×	×
Add with Carry to Accumulator	ADDC A,#data	0	0	1	1	0	1	0	0	2	1	A ← A + C + #data	×	×	×
	ADDC A,direct	0	0	1	1	0	1	0	1	2	1	A ← A + C + (direct)	×	×	×
	ADDC A,@Ri	0	0	1	1	0	1	1	**i**	1	1	A ← A + C + (Ri)	×	×	×
	ADDC A,Rn	0	0	1	1	1	**r**	**r**	**r**	1	1	A ← A + C + Rn	×	×	×
Decimal Adjust Accum. for Addition	DA A	1	1	0	1	0	1	0	0	1	1	Decimal Adjust Accum. for BCD Addition	×		
Decrement	DEC A	0	0	0	1	0	1	0	0	1	1	A ← A - 1 → A			
	DEC direct	0	0	0	1	0	1	0	1	2	1	(direct) ← (direct) - 1			
	DEC @Ri	0	0	0	1	0	1	1	**i**	1	1	(Ri) ← (Ri) - 1			
	DEC Rn	0	0	0	1	1	**r**	**r**	**r**	1	1	Rn ← Rn - 1			
Divide Accumulator by B reg.	DIV AB	1	0	0	0	0	1	0	0	1	4	A ← int(A/B) , B ← mod(A/B)	**0**		×
Increment	INC A	0	0	0	0	0	1	0	0	1	1	A ← A + 1 → A			
	INC direct	0	0	0	0	0	1	0	1	2	1	(direct) ← (direct) + 1			
	INC @Ri	0	0	0	0	0	1	1	**i**	1	1	(Ri) ← (Ri) + 1			
	INC Rn	0	0	0	0	1	**r**	**r**	**r**	1	1	Rn ← Rn + 1			
	INC DPTR	1	0	1	0	0	0	1	1	1	2	DPTR ← DPTR + 1			
Multiply Accumulator by B reg.	MUL AB	1	0	1	0	0	1	0	0	1	4	B:A ← A x B	**0**		×
Subtract with Borrow from the Accumulator	SUBB A,#data	1	0	0	1	0	1	0	0	2	1	A ← A - C - #data	×	×	×
	SUBB A,direct	1	0	0	1	0	1	0	1	2	1	A ← A - C - (direct)	×	×	×
	SUBB A,@Ri	1	0	0	1	0	1	1	**i**	1	1	A ← A - C - (Ri)	×	×	×
	SUBB A,Rn	1	0	0	1	1	**r**	**r**	**r**	1	1	A ← A - C - Rn	×	×	×

Logical Operations

Function	Instruction	OpCode/Byte/Cycle										Operation	PSW		
		b7	b6	b5	b4	b3	b2	b1	b0	#	~		CY	AC	OV
Logical AND	ANL direct,A	0	1	0	1	0	0	1	0	2	1	(direct) ← (direct) · A			
	ANL direct,#data	0	1	0	1	0	0	1	1	3	2	(direct) ← (direct) · #data			
	ANL A,#data	0	1	0	1	0	1	0	0	2	1	A ← A · #data			
	ANL A,direct	0	1	0	1	0	1	0	1	2	1	A ← A · (direct)			
	ANL A,@Ri	0	1	0	1	0	1	1	**i**	1	1	A ← A · (Ri)			
	ANL A,Rn	0	1	0	1	1	**r**	**r**	**r**	1	1	A ← A · Rn			
Logical Inclusive OR	ORL direct,A	0	1	0	0	0	0	1	0	2	1	(direct) ← (direct) + A			
	ORL direct,#data	0	1	0	0	0	0	1	1	3	2	(direct) ← (direct) + #data			
	ORL A,#data	0	1	0	0	0	1	0	0	2	1	A ← A + #data			
	ORL A,direct	0	1	0	0	0	1	0	1	2	1	A ← A + (direct)			
	ORL A,@Ri	0	1	0	0	0	1	1	**i**	1	1	A ← A + (Ri)			
	ORL A,Rn	0	1	0	0	1	**r**	**r**	**r**	1	1	A ← A + Rn			
Logical Exclusive OR	XRL direct,A	0	1	1	0	0	0	1	0	2	1	(direct) ← (direct) ⊕ A			
	XRL direct,#data	0	1	1	0	0	0	1	1	3	2	(direct) ← (direct) ⊕ #data			
	XRL A,#data	0	1	1	0	0	1	0	0	2	1	A ← A ⊕ #data			
	XRL A,direct	0	1	1	0	0	1	0	1	2	1	A ← A ⊕ (direct)			
	XRL A,@Ri	0	1	1	0	0	1	1	**i**	1	1	A ← A ⊕ (Ri)			
	XRL A,Rn	0	1	1	0	1	**r**	**r**	**r**	1	1	A ← A ⊕ Rn			
Clear Accumulator	CLR A	1	1	1	0	0	1	0	0	1	1	A ← 00			
Complement Accumulator	CPL A	1	1	1	1	0	1	0	0	1	1	A ← A'			
Rotate A Left One Bit	RL A	0	0	1	0	0	0	1	1	1	1	$A_{n+1} \leftarrow A_n$, $A_0 \leftarrow A_7$			
Rotate A Left One Bit Thru the CY	RLC A	0	0	1	1	0	0	1	1	1	1	$A_0 \leftarrow C$, $A_{n+1} \leftarrow A_n$, $C \leftarrow A_7$	×		
Rotate A Right One Bit	RR A	0	0	0	0	0	0	1	1	1	1	$A_n \leftarrow A_{n+1}$, $A_7 \leftarrow A_0$			
Rotate A Right One Bit Thru the CY	RRC A	0	0	0	1	0	0	1	1	1	1	$A_7 \leftarrow C$, $A_n \leftarrow A_{n+1}$, $C \leftarrow A_0$	×		
Swap Nibbles within the Accum.	SWAP A	1	1	0	0	0	1	0	0	1	1	$A_{3}:A_0 \leftrightarrow A_7:A_4$			

Data Transfer

Function	Instruction	OpCode/Byte/Cycle										Operation	PSW		
		b7	b6	b5	b4	b3	b2	b1	b0	#	~		CY	AC	OV
Move Source to Destination	MOV A,#data	0	1	1	1	0	1	0	0	2	1	A ← #data			
	MOV A,direct	1	1	1	0	0	1	0	1	2	1	A ← (direct)			
	MOV A,@Ri	1	1	1	0	0	1	1	**i**	1	1	A ← (Ri)			
	MOV A,Rn	1	1	1	0	1	**r**	**r**	**r**	1	1	A ← Rn			
	MOV direct,A	1	1	1	1	0	1	0	1	2	1	(direct) ← A			
	MOV direct,#data	0	1	1	1	0	1	0	1	3	2	(direct) ← #data			
	MOV direct,direct	1	0	0	0	0	1	0	1	3	2	(direct) ← (direct)			
	MOV direct,@Ri	1	0	0	0	0	1	1	**i**	2	2	(direct) ← (Ri)			
	MOV direct,Rn	1	0	0	0	1	**r**	**r**	**r**	2	2	(direct) ← Rn			
	MOV @Ri,A	1	1	1	1	0	1	1	**i**	1	1	(Ri) ← A			
	MOV @Ri,#data	0	1	1	1	0	1	1	**i**	2	1	(Ri) ← #data			
	MOV @Ri,direct	1	0	1	0	0	1	1	**i**	2	2	(Ri) ← (direct)			
	MOV Rn,A	1	1	1	1	1	**r**	**r**	**r**	1	1	Rn ← A			
	MOV Rn,#data	0	1	1	1	1	**r**	**r**	**r**	2	1	Rn ← #data			
	MOV Rn,direct	1	0	1	0	1	**r**	**r**	**r**	2	2	Rn ← (direct)			

Data Transfer (Continued)

Function	Instruction	OpCode/Byte/Cycle										Operation	PSW		
		b7	b6	b5	b4	b3	b2	b1	b0	#	~		CY	AC	OV
Move Source to DPTR	MOV DPTR,#data16	1	0	0	1	0	0	0	0	3	2	DPTR ← #data16			
Move byte from Destination	MOVC A,@A+DPTR	1	0	0	1	0	0	1	1	1	2	A ← (A + DPTR)			
	MOVC A,@A+PC	1	0	0	0	0	0	1	1	1	2	A ← (A + PC)			
Move byte from External Data Memory to the Accumulator	MOVX A,@Ri	1	1	1	0	0	0	1	**i**	1	2	A ← (Ri)			
	MOVX A,@DPTR	1	1	1	0	0	0	0	0	1	2	A ← (DPTR)			
Move byte in the Accumulator to External Data Memory	MOVX @Ri,A	1	1	1	1	0	0	1	**i**	1	2	A ← (Ri)			
	MOVX @DPTR,A	1	1	1	1	0	0	0	0	1	2	(DPTR) ← A			
Pop Stack and Place in Dest.	POP direct	1	1	0	1	0	0	0	0	2	2	(direct) ← (SP) , SP ← SP-1			
Push Source on to Stack	PUSH direct	1	1	0	0	0	0	0	0	2	2	SP ← SP+1 , (SP) ← (direct)			
Exchange bytes of the Accum. and the Source	XCH A,direct	1	1	0	0	0	1	0	1	2	1	A ↔ (direct)			
	XCH A,@Ri	1	1	0	0	0	1	1	**i**	1	1	A ↔ (Ri)			
	XCH A,Rn	1	1	0	0	1	**r**	**r**	**r**	1	1	A ↔ Rn			
Exchange digit	XCHD A,@Ri	1	1	0	1	0	1	1	**i**	1	1	A (b3:b0) ↔ Ri (b3:b0)			

Boolean Variable Manipulation

Function	Instruction	OpCode/Byte/Cycle										Operation	PSW		
		b7	b6	b5	b4	b3	b2	b1	b0	#	~		CY	AC	OV
Clear Bit Operand	CLR C	1	1	0	0	0	0	1	1	1	1	C ← 0	**0**		
	CLR bit	1	1	0	0	0	0	1	0	2	1	bit ← 0			
Set Bit Operand	SETB C	1	1	0	1	0	0	1	1	1	1	C ← 0	**1**		
	SETB bit	1	1	0	1	0	0	1	0	2	1	bit ← 1			
Complement Carry Flag	CPL C	1	0	1	1	0	0	1	1	1	1	C ← C'	×		
Complement Bit Operand	CPL bit	1	0	1	1	0	0	1	0	2	1	bit → bit'			
Logical AND Bit Operand	ANL C,bit	1	0	0	0	0	0	1	0	2	2	C ← C · bit	×		
	ANL C,/bit	1	0	1	1	0	0	0	0	2	2	C ← C · bit'	×		
Logical Inclusive OR Bit Operand	ORL C,bit	0	1	1	1	0	0	1	0	2	2	C ← C + bit	×		
	ORL C,/bit	1	0	1	0	0	0	0	0	2	2	C ← C + bit'	×		
Move Carry Flag to Bit	MOV bit,C	1	0	0	1	0	0	1	0	2	2	bit ← C	×		
Move Bit to Carry Flag	MOV C,bit	1	0	1	0	0	0	1	0	2	1	C ← bit	×		
Jump Relative if Carry Flag is Set	JC rel	0	1	0	0	0	0	0	0	2	2	PC←PC+2, if C=1 then PC←PC+rel			
Jump Relative if Carry Flag is Clear	JNC rel	0	1	0	1	0	0	0	0	2	2	PC←PC+2, if C=0 then PC←PC+rel			
Jump Relative if bit is Set	JB bit,rel	0	0	1	0	0	0	0	0	3	2	PC←PC+3, if bit=1 then PC←PC+rel			
Jump Relative if bit is Clear	JNB bit,rel	0	0	1	1	0	0	0	0	3	2	PC←PC+3, if bit=0 then PC←PC+rel			
Jump Relative if bit is Set and Clear bit	JBC bit,rel	0	0	0	1	0	0	0	0	3	2	PC←PC+3, if bit=1 then bit←0 , PC←PC+rel			

Program Branching

Function	Instruction	OpCode/Byte/Cycle										Operation	PSW		
		b7	b6	b5	b4	b3	b2	b1	b0	#	~		CY	AC	OV
Absolute Call , 2K in Page (11 bits)	ACALL addr11	**a10**	**a9**	**a8**	1	0	0	0	1	2	2	PC←PC+2, SP←SP+1, (SP)←PCL, SP←SP+1, (SP)←PCH, PC←addr11			
Long (16 bits) Call	LCALL addr16	0	0	0	1	0	0	1	0	3	2	PC←PC+6, SP←SP+1, (SP)←PCL, SP←SP+1, (SP)←PCH, PC←addr16			
Return from Subroutine	RET	0	0	1	0	0	0	1	0	1	2	PCH←(SP), SP←SP-1, PCL←(SP), SP←SP-1			
Return from Interrupt Routine	RETI	0	0	1	1	0	0	1	0	1	2	PCH←(SP), SP←SP-1, PCL←(SP), SP←SP-1			
2K in Page (11 bits) Absolute Jump	AJMP addr11	**a10**	**a9**	**a8**	0	0	0	0	1	2	2	PC←addr11			
Long (16 bits) Absolute Jump	LJMP addr16	0	0	0	0	0	0	1	0	3	2	PC←addr16			
Short (8 bits) Relative Jump	SJMP rel	1	0	0	0	0	0	0	0	2	2	PC←PC+2, PC←PC+rel			
Jump Indirect	JMP @A+DPTR	0	1	1	1	0	0	1	1	1	2	PC←A+DPTR			
Jump Relative if the Accum is Zero	JZ rel	0	1	1	0	0	0	0	0	2	2	PC←PC+2, if A=0 then PC←PC+rel			
Jump Relative if the Accum is not Zero	JNZ rel	0	1	1	1	0	0	0	0	2	2	PC←PC+2, if A≠0 then PC←PC+rel			
Compare Operands and Jump Relative if not Equal	CJNE A,#data,rel	1	0	1	1	0	1	0	0	3	2	PC←PC+3, if A ≠ #data then PC←PC+rel, if A<#data then C←1 else C←0	×		
	CJNE A,direct,rel	1	0	1	1	0	1	0	1	3	2	PC←PC+3, if A ≠ (direct) then PC←PC+rel, if A<(direct) then C←1 else C←0	×		
	CJNE @Ri,#data,rel	1	0	1	1	0	1	1	**i**	3	2	PC←PC+3, if (Ri) ≠ #data then PC←PC+rel, if (Ri)<#data then C←1 else C←0	×		
	CJNE Rn,#data,rel	1	0	1	1	1	**r**	**r**	**r**	3	2	PC←PC+3, if Rn ≠ #data then PC←PC+rel, if Rn<#data then C←1 else C←0	×		
Decrement Operand and Jump Relative if Not Zero	DJNZ direct,rel	1	1	0	1	0	1	0	1	3	2	PC←PC+2, (direct)← (direct)-1, if (direct)>0 or (direct)<0 then PC←PC+rel			
	DJNZ Rn,rel	1	1	0	1	1	**r**	**r**	**r**	2	2	PC←PC+2, Rn ← Rn-1, if Rn>0 or Rn<0 then PC←PC+rel			
No Operation	NOP	0	0	0	0	0	0	0	0	1	1	only increment PC			

A : Accumulator **PC** : Program Counter **SP** : Stack Pointer **DPTR** : Data Pointer **PSW** : Program Status Word [CY: Cary AC: Aux.Cary OV: Overflow]

Notes on Data Addressing Modes

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

Notes on Program Addressing Modes

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