

INSTRUCTION SET

OVERVIEW

The SAM87RC instruction set is specifically designed to support the large register files that are typical of most SAM87RC microcontrollers. There are 78 instructions. The powerful data manipulation capabilities and features of the instruction set include:

- A full complement of 8-bit arithmetic and logic operations, including multiply and divide
- No special I/O instructions (I/O control/data registers are mapped directly into the register file)
- Decimal adjustment included in binary-coded decimal (BCD) operations
- 16-bit (word) data can be incremented and decremented
- Flexible instructions for bit addressing, rotate, and shift operations

DATA TYPES

The SAM87RC CPU performs operations on bits, bytes, BCD digits, and two-byte words. Bits in the register file can be set, cleared, complemented, and tested. Bits within a byte are numbered from 7 to 0, where bit 0 is the least significant (right-most) bit.

REGISTER ADDRESSING

To access an individual register, an 8-bit address in the range 0-255 or the 4-bit address of a working register is specified. Paired registers can be used to construct 16-bit data or 16-bit program memory or data memory addresses. For detailed information about register addressing, please refer to Section 2, "Address Spaces."

ADDRESSING MODES

There are seven explicit addressing modes: Register (R), Indirect Register (IR), Indexed (X), Direct (DA), Relative (RA), Immediate (IM), and Indirect (IA). For detailed descriptions of these addressing modes, please refer to Section 3, "Addressing Modes."

Table 6-1. Instruction Group Summary

Mnemonic	Operands	Instruction
Load Instructions		
CLR	dst	Clear
LD	dst,src	Load
LDB	dst,src	Load bit
LDE	dst,src	Load external data memory
LDC	dst,src	Load program memory
LDED	dst,src	Load external data memory and decrement
LDCD	dst,src	Load program memory and decrement
LDEI	dst,src	Load external data memory and increment
LDCI	dst,src	Load program memory and increment
LDEPD	dst,src	Load external data memory with pre-decrement
LDCPD	dst,src	Load program memory with pre-decrement
LDEPI	dst,src	Load external data memory with pre-increment
LDCPI	dst,src	Load program memory with pre-increment
LDW	dst,src	Load word
POP	dst	Pop from stack
POPUD	dst,src	Pop user stack (decrementing)
POPUI	dst,src	Pop user stack (incrementing)
PUSH	src	Push to stack
PUSHUD	dst,src	Push user stack (decrementing)
PUSHUI	dst,src	Push user stack (incrementing)



Table 6-1. Instruction Group Summary (Continued)

Operands	Instruction
ons	
dst,src	Add with carry
dst,src	Add
dst,src	Compare
dst	Decimal adjust
dst	Decrement
dst	Decrement word
dst,src	Divide
dst	Increment
dst	Increment word
dst,src	Multiply
dst,src	Subtract with carry
dst,src	Subtract
dst,src	Logical AND
dst	Complement
dst,src	Logical OR
dst,src	Logical exclusive OR
	dst,src dst,src dst



Table 6-1. Instruction Group Summary (Continued)

Mnemonic	Operands	Instruction	_				
Program Control Ir	nstructions						
BTJRF	dst,src	Bit test and jump relative on false					
BTJRT	dst,src	Bit test and jump relative on true					
CALL	dst	Call procedure					
CPIJE	dst,src	Compare, increment and jump on equal					
CPIJNE	dst,src	Compare, increment and jump on non-equal					
DJNZ	r,dst	Decrement register and jump on non-zero					
ENTER		Enter					
EXIT		Exit					
IRET		Interrupt return					
JP	cc,dst	Jump on condition code					
JP	dst	Jump unconditional					
JR	cc,dst	Jump relative on condition code					
NEXT		Next					
RET		Return					
WFI		Wait for interrupt					
Bit Manipulation In	structions						
BAND	dst,src	Bit AND					
BCP	dst,src	Bit compare					
BITC	dst	Bit complement					
BITR	dst	Bit reset					
BITS	dst	Bit set					
BOR	dst,src	Bit OR					
BXOR	dst,src	Bit XOR					

Test complement under mask

Test under mask



TCM

TM

dst,src

dst,src

Table 6-1. Instruction Group Summary (Concluded)

Mnemon	ic Operands	Instruction	
Rotate and Sh	ift Instructions		
RL	dst	Rotate left	
RLC	dst	Rotate left through carry	
RR	dst	Rotate right	

Swap nibbles

Rotate right through carry

Shift right arithmetic

CPU Control Instructions

dst

dst

dst

RRC

SRA

SWAP

CCF		Complement carry flag
DI		Disable interrupts
El		Enable interrupts
IDLE		Enter Idle mode
NOP		No operation
RCF		Reset carry flag
SB0		Set bank 0
SB1		Set bank 1
SCF		Set carry flag
SRP	src	Set register pointers
SRP0	src	Set register pointer 0
SRP1	src	Set register pointer 1
STOP		Enter Stop mode
= =		



FLAGS REGISTER (FLAGS)

The flags register FLAGS contains eight bits that describe the current status of CPU operations. Four of these bits, FLAGS.7–FLAGS.4, can be tested and used with conditional jump instructions; two others FLAGS.3 and FLAGS.2 are used for BCD arithmetic.

The FLAGS register also contains a bit to indicate the status of fast interrupt processing (FLAGS.1) and a bank address status bit (FLAGS.0) to indicate whether bank 0 or bank 1 is currently being addressed. FLAGS register can be set or reset by instructions as long as its outcome does not affect the flags, such as, Load instruction.

Logical and Arithmetic instructions such as, AND, OR, XOR, ADD, and SUB can affect the Flags register. For example, the AND instruction updates the Zero, Sign and Overflow flags based on the outcome of the AND instruction. If the AND instruction uses the Flags register as the destination, then simultaneously, two write will occur to the Flags register producing an unpredictable result.

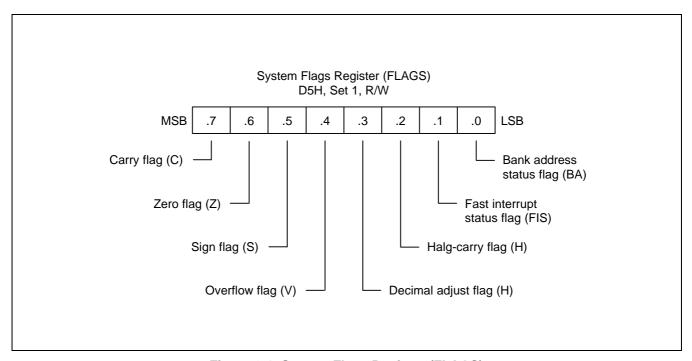


Figure 6-1. System Flags Register (FLAGS)

FLAG DESCRIPTIONS

C Carry Flag (FLAGS.7)

The C flag is set to "1" if the result from an arithmetic operation generates a carry-out from or a borrow to the bit 7 position (MSB). After rotate and shift operations, it contains the last value shifted out of the specified register. Program instructions can set, clear, or complement the carry flag.

Z Zero Flag (FLAGS.6)

For arithmetic and logic operations, the Z flag is set to "1" if the result of the operation is zero. For operations that test register bits, and for shift and rotate operations, the Z flag is set to "1" if the result is logic zero.

Sign Flag (FLAGS.5)

Following arithmetic, logic, rotate, or shift operations, the sign bit identifies the state of the MSB of the result. A logic zero indicates a positive number and a logic one indicates a negative number.

V Overflow Flag (FLAGS.4)

The V flag is set to "1" when the result of a two's-complement operation is greater than + 127 or less than – 128. It is also cleared to "0" following logic operations.

D Decimal Adjust Flag (FLAGS.3)

The DA bit is used to specify what type of instruction was executed last during BCD operations, so that a subsequent decimal adjust operation can execute correctly. The DA bit is not usually accessed by programmers, and cannot be used as a test condition.

Half-Carry Flag (FLAGS.2)

The H bit is set to "1" whenever an addition generates a carry-out of bit 3, or when a subtraction borrows out of bit 4. It is used by the Decimal Adjust (DA) instruction to convert the binary result of a previous addition or subtraction into the correct decimal (BCD) result. The H flag is seldom accessed directly by a program.

FIS Fast Interrupt Status Flag (FLAGS.1)

The FIS bit is set during a fast interrupt cycle and reset during the IRET following interrupt servicing. When set, it inhibits all interrupts and causes the fast interrupt return to be executed when the IRET instruction is executed.

BA Bank Address Flag (FLAGS.0)

The BA flag indicates which register bank in the set 1 area of the internal register file is currently selected, bank 0 or bank 1. The BA flag is cleared to "0" (select bank 0) when you execute the SB0 instruction and is set to "1" (select bank 1) when you execute the SB1 instruction.

INSTRUCTION SET NOTATION

Table 6-2. Flag Notation Conventions

Flag	Description
С	Carry flag
Z	Zero flag
S	Sign flag
V	Overflow flag
D	Decimal-adjust flag
Н	Half-carry flag
0	Cleared to logic zero
1	Set to logic one
*	Set or cleared according to operation
_	Value is unaffected
x	Value is undefined

Table 6-3. Instruction Set Symbols

Symbol	Description
dst	Destination operand
src	Source operand
@	Indirect register address prefix
PC	Program counter
IP	Instruction pointer
FLAGS	Flags register (D5H)
RP	Register pointer
#	Immediate operand or register address prefix
Н	Hexadecimal number suffix
D	Decimal number suffix
В	Binary number suffix
орс	Opcode

Table 6-4. Instruction Notation Conventions

Notation	Description	Actual Operand Range			
СС	Condition code	See list of condition codes in Table 6-6.			
r	Working register only	Rn (n = 0–15)			
rb	Bit (b) of working register	Rn.b (n = 0–15, b = 0–7)			
r0	Bit 0 (LSB) of working register	Rn (n = 0–15)			
rr	Working register pair	RRp (p = 0, 2, 4,, 14)			
R	Register or working register	reg or Rn (reg = 0-255, n = 0-15)			
Rb	Bit 'b' of register or working register	reg.b (reg = 0–255, b = 0–7)			
RR	Register pair or working register pair	reg or RRp (reg = 0–254, even number only, where $p = 0, 2,, 14$)			
IA	Indirect addressing mode	addr (addr = 0–254, even number only)			
Ir	Indirect working register only	@Rn (n = 0–15)			
IR	Indirect register or indirect working register	@Rn or @reg (reg = 0-255, n = 0-15)			
Irr	Indirect working register pair only	@RRp (p = 0, 2,, 14)			
IRR	Indirect register pair or indirect working register pair	@RRp or @reg (reg = 0–254, even only, where p = 0, 2,, 14)			
Х	Indexed addressing mode	#reg [Rn] (reg = 0–255, n = 0–15)			
xs	Indexed (short offset) addressing mode	#addr [RRp] (addr = range -128 to +127, where p = 0, 2,, 14)			
XL	Indexed (long offset) addressing mode	#addr [RRp] (addr = range 0–65535, where p = 0, 2,, 14)			
DA	Direct addressing mode	addr (addr = range 0-65535)			
RA	Relative addressing mode	addr (addr = number in the range +127 to -128 that is an offset relative to the address of the next instruction)			
IM	Immediate addressing mode	#data (data = 0-255)			
IML	Immediate (long) addressing mode	#data (data = range 0-65535)			

Table 6-5. Opcode Quick Reference

	OPCODE MAP									
				LOWER	NIBBLE (F	IEX)				
	-	0	1	2	3	4	5	6	7	
U	0	DEC R1	DEC IR1	ADD r1,r2	ADD r1,lr2	ADD R2,R1	ADD IR2,R1	ADD R1,IM	BOR r0–Rb	
Р	1	RLC R1	RLC IR1	ADC r1,r2	ADC r1,lr2	ADC R2,R1	ADC IR2,R1	ADC R1,IM	BCP r1.b, R2	
Р	2	INC R1	INC IR1	SUB r1,r2	SUB r1,lr2	SUB R2,R1	SUB IR2,R1	SUB R1,IM	BXOR r0–Rb	
E	3	JP IRR1	SRP/0/1 IM	SBC r1,r2	SBC r1,lr2	SBC R2,R1	SBC IR2,R1	SBC R1,IM	BTJR r2.b, RA	
R	4	DA R1	DA IR1	OR r1,r2	OR r1,lr2	OR R2,R1	OR IR2,R1	OR R1,IM	LDB r0–Rb	
	5	POP R1	POP IR1	AND r1,r2	AND r1,lr2	AND R2,R1	AND IR2,R1	AND R1,IM	BITC r1.b	
N	6	COM R1	COM IR1	TCM r1,r2	TCM r1,lr2	TCM R2,R1	TCM IR2,R1	TCM R1,IM	BAND r0–Rb	
I	7	PUSH R2	PUSH IR2	TM r1,r2	TM r1,lr2	TM R2,R1	TM IR2,R1	TM R1,IM	BIT r1.b	
В	8	DECW RR1	DECW IR1	PUSHUD IR1,R2	PUSHUI IR1,R2	MULT R2,RR1	MULT IR2,RR1	MULT IM,RR1	LD r1, x, r2	
В	9	RL R1	RL IR1	POPUD IR2,R1	POPUI IR2,R1	DIV R2,RR1	DIV IR2,RR1	DIV IM,RR1	LD r2, x, r1	
L	Α	INCW RR1	INCW IR1	CP r1,r2	CP r1,lr2	CP R2,R1	CP IR2,R1	CP R1,IM	LDC r1, lrr2, xL	
E			CLR IR1	XOR r1,r2	XOR r1,lr2	XOR R2,R1	XOR IR2,R1	XOR R1,IM	LDC r2, lrr2, xL	
	С	RRC R1	RRC IR1	CPIJE Ir,r2,RA	LDC r1,lrr2	LDW RR2,RR1	LDW IR2,RR1	LDW RR1,IML	LD r1, lr2	
Н	D	SRA R1	SRA IR1	CPIJNE Irr,r2,RA	LDC r2,lrr1	CALL IA1		LD IR1,IM	LD Ir1, r2	
E	Е	RR R1	RR IR1	LDCD r1,lrr2	LDCI r1,lrr2	LD R2,R1	LD R2,IR1	LD R1,IM	LDC r1, Irr2, xs	
X	F	SWAP R1	SWAP IR1	LDCPD r2,lrr1	LDCPI r2,Irr1	CALL IRR1	LD IR2,R1	CALL DA1	LDC r2, lrr1, xs	

Table 6-5. Opcode Quick Reference (Continued)

	OPCODE MAP										
				LOWER	NIBBLE (H	IEX)					
	_	8	9	А	В	С	D	Е	F		
C	0	LD r1,R2	LD r2,R1	DJNZ r1,RA	JR cc,RA	LD r1,IM	JP cc,DA	INC r1	NEXT		
Р	1	\	\downarrow	\downarrow	\downarrow	\	\downarrow	\downarrow	ENTER		
Р	2								EXIT		
E	3								WFI		
R	4								SB0		
	5								SB1		
N	6								IDLE		
ı	7	\	\downarrow	\downarrow	\downarrow	\	\downarrow	\downarrow	STOP		
В	8								DI		
В	9								EI		
L	Α								RET		
E	В								IRET		
	С								RCF		
Н	D	\	\downarrow	\downarrow	\downarrow	\	\downarrow	\downarrow	SCF		
E	E								CCF		
Х	F	LD r1,R2	LD r2,R1	DJNZ r1,RA	JR cc,RA	LD r1,IM	JP cc,DA	INC r1	NOP		

CONDITION CODES

The opcode of a conditional jump always contains a 4-bit field called the condition code (cc). This specifies under which conditions it is to execute the jump. For example, a conditional jump with the condition code for "equal" after a compare operation only jumps if the two operands are equal. Condition codes are listed in Table 6-6.

The carry (C), zero (Z), sign (S), and overflow (V) flags are used to control the operation of conditional jump instructions.

Table 6-6. Condition Codes

Binary	Mnemonic	Description	Flags Set		
0000	F	Always false	-		
1000	Т	Always true	-		
0111 ^(note)	С	Carry C = 1			
1111 ^(note)	NC	No carry	C = 0		
0110 ^(note)	z	Zero	Z = 1		
1110 ^(note)	NZ	Not zero	Z = 0		
1101	PL	Plus	S = 0		
0101	МІ	Minus	S = 1		
0100	ov	Overflow	V = 1		
1100	NOV	No overflow	V = 0		
0110 ^(note)	EQ	Equal	Z = 1		
1110 ^(note)	NE	Not equal	Z = 0		
1001	GE	Greater than or equal	(S XOR V) = 0		
0001	LT	Less than	(S XOR V) = 1		
1010	GT	Greater than	(Z OR (S XOR V)) = 0		
0010	LE	Less than or equal	(Z OR (S XOR V)) = 1		
1111 ^(note)	UGE	Unsigned greater than or equal	C = 0		
0111 ^(note)	ULT	Unsigned less than	C = 1		
1011	UGT	Unsigned greater than	(C = 0 AND Z = 0) = 1		
0011	ULE	Unsigned less than or equal	(C OR Z) = 1		

NOTES:

- 1. It indicates condition codes that are related to two different mnemonics but which test the same flag. For example, Z and EQ are both true if the zero flag (Z) is set, but after an ADD instruction, Z would probably be used; after a CP instruction, however, EQ would probably be used.
- 2. For operations involving unsigned numbers, the special condition codes UGE, ULT, UGT, and ULE must be used.



INSTRUCTION DESCRIPTIONS

This section contains detailed information and programming examples for each instruction in the SAM87RC instruction set. Information is arranged in a consistent format for improved readability and for fast referencing. The following information is included in each instruction description:

- Instruction name (mnemonic)
- Full instruction name
- Source/destination format of the instruction operand
- Shorthand notation of the instruction's operation
- Textual description of the instruction's effect
- Specific flag settings affected by the instruction
- Detailed description of the instruction's format, execution time, and addressing mode(s)
- Programming example(s) explaining how to use the instruction



ADC — Add with carry

ADC dst,src

Operation: $dst \leftarrow dst + src + c$

The source operand, along with the setting of the carry flag, is added to the destination operand and the sum is stored in the destination. The contents of the source are unaffected. Two's-complement addition is performed. In multiple precision arithmetic, this instruction permits the carry from the addition of low-order operands to be carried into the addition of high-order operands.

Flags: C: Set if there is a carry from the most significant bit of the result; cleared otherwise.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result is negative; cleared otherwise.

V: Set if arithmetic overflow occurs, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise.

D: Always cleared to "0".

H: Set if there is a carry from the most significant bit of the low-order four bits of the result; cleared otherwise.

Format:

				Bytes	Cycles	Opcode (Hex)	Addr <u>dst</u>	Mode <u>src</u>
орс	dst src			2	4	12	r	r
					6	13	r	lr
			,					
орс	src	dst		3	6	14	R	R
					6	15	R	IR
орс	dst	src		3	6	16	R	IM

Examples:

Given: R1 = 10H, R2 = 03H, C flag = "1", register 01H = 20H, register 02H = 03H, and register 03H = 0AH:

ADC R1,R2
$$\rightarrow$$
 R1 = 14H, R2 = 03H
ADC R1,@R2 \rightarrow R1 = 1BH, R2 = 03H
ADC 01H,02H \rightarrow Register 01H = 24H, register 02H = 03H
ADC 01H,@02H \rightarrow Register 01H = 2BH, register 02H = 03H
ADC 01H,#11H \rightarrow Register 01H = 32H

In the first example, destination register R1 contains the value 10H, the carry flag is set to "1", and the source working register R2 contains the value 03H. The statement "ADC R1,R2" adds 03H and the carry flag value ("1") to the destination value 10H, leaving 14H in register R1.



ADD — Add

ADD dst,src

Operation: $dst \leftarrow dst + src$

The source operand is added to the destination operand and the sum is stored in the destination.

The contents of the source are unaffected. Two's-complement addition is performed.

Flags: C: Set if there is a carry from the most significant bit of the result; cleared otherwise.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result is negative; cleared otherwise.

V: Set if arithmetic overflow occurred, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise.

D: Always cleared to "0".

H: Set if a carry from the low-order nibble occurred.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr <u>dst</u>	Mode <u>src</u>
орс	dst src		2	4	02	r	r
				6	03	r	lr
орс	src	dst	3	6	04	R	R
				6	05	R	IR
орс	dst	src	3	6	06	R	IM

Examples: Given: R1 = 12H, R2 = 03H, register R1 = 12H, registe

ADD R1,R2 R1 = 15H, R2 = 03HADD R1 = 1CH, R2 = 03HR1,@R2 ADD Register 01H = 24H, register 02H = 03H 01H,02H \rightarrow ADD 01H,@02H Register 01H = 2BH, register 02H = 03H ADD Register 01H = 46H 01H.#25H

In the first example, destination working register R1 contains 12H and the source working register R2 contains 03H. The statement "ADD R1,R2" adds 03H to 12H, leaving the value 15H in register R1.

AND — Logical AND

AND dst,src

Operation: $dst \leftarrow dst \ AND \ src$

The source operand is logically ANDed with the destination operand. The result is stored in the destination. The AND operation results in a "1" bit being stored whenever the corresponding bits in the two operands are both logic ones; otherwise a "0" bit value is stored. The contents of the

source are unaffected.

Flags: C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result bit 7 is set; cleared otherwise.

V: Always cleared to "0".

D: Unaffected.H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr <u>dst</u>	Mode <u>src</u>
орс	dst src		2	4	52	r	r
				6	53	r	lr
орс	src	dst	3	6	54	R	R
				6	55	R	IR
орс	dst	src	3	6	56	R	IM

Examples: Given: R1 = 12H, R2 = 03H, register 01H = 21H, register 02H = 03H, register 03H = 0AH:

AND R1,R2
$$\rightarrow$$
 R1 = 02H, R2 = 03H
AND R1,@R2 \rightarrow R1 = 02H, R2 = 03H
AND 01H,02H \rightarrow Register 01H = 01H, register 02H = 03H
AND 01H,@02H \rightarrow Register 01H = 00H, register 02H = 03H
AND 01H,#25H \rightarrow Register 01H = 21H

In the first example, destination working register R1 contains the value 12H and the source working register R2 contains 03H. The statement "AND R1,R2" logically ANDs the source operand 03H with the destination operand value 12H, leaving the value 02H in register R1.



BAND — Bit AND

BAND dst,src.b

BAND dst.b,src

Operation: $dst(0) \leftarrow dst(0)$ AND src(b)

or

 $dst(b) \leftarrow dst(b) AND src(0)$

The specified bit of the source (or the destination) is logically ANDed with the zero bit (LSB) of the destination (or source). The resultant bit is stored in the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

Flags: C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Cleared to "0".V: Undefined.D: Unaffected.H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr <u>dst</u>	Mode <u>src</u>
орс	dst b 0	src	3	6	67	rO	Rb
opc	src b 1	dst	3	6	67	Rb	r0

NOTE: In the second byte of the 3-byte instruction formats, the destination (or source) address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Examples: Given: R1 = 07H and register 01H = 05H:

BAND R1,01H.1 \rightarrow R1 = 06H, register 01H = 05H BAND 01H.1,R1 \rightarrow Register 01H = 05H, R1 = 07H

In the first example, source register 01H contains the value 05H (00000101B) and destination working register R1 contains 07H (00000111B). The statement "BAND R1,01H.1" ANDs the bit 1 value of the source register ("0") with the bit 0 value of register R1 (destination), leaving the value 06H (00000110B) in register R1.

BCP — Bit Compare

BCP dst,src.b

Operation: dst(0) - src(b)

The specified bit of the source is compared to (subtracted from) bit zero (LSB) of the destination.

The zero flag is set if the bits are the same; otherwise it is cleared. The contents of both

operands are unaffected by the comparison.

Flags: C: Unaffected

Z: Set if the two bits are the same; cleared otherwise.

S: Cleared to "0".V: Undefined.D: Unaffected.H: Unaffected.

Format:

			Bytes	Cycles	Opcode	Addr	Mode
					(Hex)	<u>dst</u>	src
орс	dst b 0	src	3	6	17	r0	Rb

NOTE: In the second byte of the instruction format, the destination address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H and register 01H = 01H:

BCP R1,01H.1 \rightarrow R1 = 07H, register 01H = 01H

If destination working register R1 contains the value 07H (00000111B) and the source register 01H contains the value 01H (00000001B), the statement "BCP R1,01H.1" compares bit one of the source register (01H) and bit zero of the destination register (R1). Because the bit values are not identical, the zero flag bit (Z) is cleared in the FLAGS register (0D5H).



BITC — Bit Complement

BITC dst.b

Operation: $dst(b) \leftarrow NOT dst(b)$

This instruction complements the specified bit within the destination without affecting any other

bits in the destination.

C: Unaffected. Flags:

Z: Set if the result is "0"; cleared otherwise.

S: Cleared to "0". V: Undefined. D: Unaffected. H: Unaffected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
орс	dst b 0	2	4	57	rb

NOTE: In the second byte of the instruction format, the destination address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H

> **BITC** R1.1 R1 = 05H

If working register R1 contains the value 07H (00000111B), the statement "BITC R1.1" complements bit one of the destination and leaves the value 05H (00000101B) in register R1. Because the result of the complement is not "0", the zero flag (Z) in the FLAGS register (0D5H) is cleared.

BITR — Bit Reset

BITR dst.b

Operation: $dst(b) \leftarrow 0$

The BITR instruction clears the specified bit within the destination without affecting any other bits

in the destination.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
opc	dst b 0	2	4	77	rb

NOTE: In the second byte of the instruction format, the destination address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H:

BITR R1.1 \rightarrow R1 = 05H

If the value of working register R1 is 07H (00000111B), the statement "BITR R1.1" clears bit one of the destination register R1, leaving the value 05H (00000101B).

BITS - Bit Set

BITS dst.b

Operation: $dst(b) \leftarrow 1$

The BITS instruction sets the specified bit within the destination without affecting any other bits

in the destination.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
орс	dst b 1	2	4	77	rb

NOTE: In the second byte of the instruction format, the destination address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H:

BITS R1.3 \rightarrow R1 = 0FH

If working register R1 contains the value 07H (00000111B), the statement "BITS R1.3" sets bit three of the destination register R1 to "1", leaving the value 0FH (00001111B).

BOR — Bit OR

BOR dst,src.b dst.b,src

Operation: $dst(0) \leftarrow dst(0)$ OR src(b)

or

 $dst(b) \leftarrow dst(b) OR src(0)$

The specified bit of the source (or the destination) is logically ORed with bit zero (LSB) of the destination (or the source). The resulting bit value is stored in the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

Flags: C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Cleared to "0".V: Undefined.D: Unaffected.H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr <u>dst</u>	Mode <u>src</u>
орс	dst b 0	src	3	6	07	r0	Rb
	,						
opc	src b 1	dst	3	6	07	Rb	r0

NOTE: In the second byte of the 3-byte instruction formats, the destination (or source) address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit.

Examples: Given: R1 = 07H and register 01H = 03H:

BOR R1, 01H.1 \rightarrow R1 = 07H, register 01H = 03H BOR 01H.2, R1 \rightarrow Register 01H = 07H, R1 = 07H

In the first example, destination working register R1 contains the value 07H (00000111B) and source register 01H the value 03H (00000011B). The statement "BOR R1,01H.1" logically ORs bit one of register 01H (source) with bit zero of R1 (destination). This leaves the same value (07H) in working register R1.

In the second example, destination register 01H contains the value 03H (00000011B) and the source working register R1 the value 07H (00000111B). The statement "BOR 01H.2,R1" logically ORs bit two of register 01H (destination) with bit zero of R1 (source). This leaves the value 07H in register 01H.



BTJRF — Bit Test, Jump Relative on False

BTJRF dst,src.b

Operation: If src(b) is a "0", then $PC \leftarrow PC + dst$

The specified bit within the source operand is tested. If it is a "0", the relative address is added to the program counter and control passes to the statement whose address is now in the PC;

otherwise, the instruction following the BTJRF instruction is executed.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode	Addr	Mode
	(Note 1)				(Hex)	<u>dst</u>	src
орс	src b 0	dst	3	10	37	RA	rb

NOTE: In the second byte of the instruction format, the source address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H:

BTJRF SKIP,R1.3 \rightarrow PC jumps to SKIP location

If working register R1 contains the value 07H (00000111B), the statement "BTJRF SKIP,R1.3" tests bit 3. Because it is "0", the relative address is added to the PC and the PC jumps to the memory location pointed to by the SKIP. (Remember that the memory location must be within the allowed range of + 127 to - 128.)

BTJRT — Bit Test, Jump Relative on True

BTJRT dst,src.b

Operation: If src(b) is a "1", then $PC \leftarrow PC + dst$

The specified bit within the source operand is tested. If it is a "1", the relative address is added to the program counter and control passes to the statement whose address is now in the PC;

otherwise, the instruction following the BTJRT instruction is executed.

Flags: No flags are affected.

Format:

			Bytes	S Cycles	Opcode	Addr	Mode
	(Note 1)				(Hex)	<u>dst</u>	src
орс	src b 1	dst	3	10	37	RA	rb

NOTE: In the second byte of the instruction format, the source address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H:

BTJRT SKIP,R1.1

If working register R1 contains the value 07H (00000111B), the statement "BTJRT SKIP,R1.1" tests bit one in the source register (R1). Because it is a "1", the relative address is added to the PC and the PC jumps to the memory location pointed to by the SKIP. (Remember that the memory location must be within the allowed range of + 127 to - 128.)

BXOR — Bit XOR

BXOR dst,src.b dst.b,src

Operation: $dst(0) \leftarrow dst(0) XOR src(b)$

or

 $dst(b) \leftarrow dst(b) XOR src(0)$

The specified bit of the source (or the destination) is logically exclusive-ORed with bit zero (LSB) of the destination (or source). The result bit is stored in the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

Flags: C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Cleared to "0".V: Undefined.D: Unaffected.H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr <u>dst</u>	Mode <u>src</u>
орс	dst b 0	src	3	6	27	r0	Rb
орс	src b 1	dst	3	6	27	Rb	r0

NOTE: In the second byte of the 3-byte instruction formats, the destination (or source) address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Examples: Given: R1 = 07H (00000111B) and register 01H = 03H (00000011B):

BXOR R1,01H.1 \rightarrow R1 = 06H, register 01H = 03H BXOR 01H.2,R1 \rightarrow Register 01H = 07H, R1 = 07H

In the first example, destination working register R1 has the value 07H (00000111B) and source register 01H has the value 03H (00000011B). The statement "BXOR R1,01H.1" exclusive-ORs bit one of register 01H (source) with bit zero of R1 (destination). The result bit value is stored in bit zero of R1, changing its value from 07H to 06H. The value of source register 01H is unaffected.

CALL — Call Procedure

CALL dst

Operation: SP \leftarrow SP – 1

 $\begin{array}{cccc} @SP & \leftarrow & PCL \\ SP & \leftarrow & SP-1 \\ @SP & \leftarrow & PCH \\ PC & \leftarrow & dst \end{array}$

The current contents of the program counter are pushed onto the top of the stack. The program counter value used is the address of the first instruction following the CALL instruction. The specified destination address is then loaded into the program counter and points to the first instruction of a procedure. At the end of the procedure the return instruction (RET) can be used to return to the original program flow. RET pops the top of the stack back into the program counter.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
opc	dst	3	14	F6	DA
opc	dst	2	12	F4	IRR
орс	dst	2	14	D4	IA

Examples: Given: R0 = 35H, R1 = 21H, PC = 1A47H, and SP = 0002H:

CALL 3521H
$$\rightarrow$$
 SP = 0000H (Memory locations 0000H = 1AH, 0001H = 4AH, where 4AH is the address that follows the instruction.)

CALL @RR0 \rightarrow SP = 0000H (0000H = 1AH, 0001H = 49H)

CALL #40H \rightarrow SP = 0000H (0000H = 1AH, 0001H = 49H)

In the first example, if the program counter value is 1A47H and the stack pointer contains the value 0002H, the statement "CALL 3521H" pushes the current PC value onto the top of the stack. The stack pointer now points to memory location 0000H. The PC is then loaded with the value 3521H, the address of the first instruction in the program sequence to be executed.

If the contents of the program counter and stack pointer are the same as in the first example, the statement "CALL @RR0" produces the same result except that the 49H is stored in stack location 0001H (because the two-byte instruction format was used). The PC is then loaded with the value 3521H, the address of the first instruction in the program sequence to be executed. Assuming that the contents of the program counter and stack pointer are the same as in the first example, if program address 0040H contains 35H and program address 0041H contains 21H, the statement "CALL #40H" produces the same result as in the second example.



CCF — Complement Carry Flag

CCF

Operation: $C \leftarrow NOT C$

The carry flag (C) is complemented. If C = "1", the value of the carry flag is changed to logic zero; if C = "0", the value of the carry flag is changed to logic one.

Flags: C: Complemented.

No other flags are affected.

Format:

Bytes Cycles Opcode (Hex) EF 1 4 opc

Example: Given: The carry flag = "0":

CCF

If the carry flag = "0", the CCF instruction complements it in the FLAGS register (0D5H), changing its value from logic zero to logic one.

CLR — Clear

CLR dst

Operation: $dst \leftarrow "0"$

The destination location is cleared to "0".

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
орс	dst	2	4	В0	R
			4	R1	IR

Examples: Given: Register 00H = 4FH, register 01H = 02H, and register 02H = 5EH:

CLR 00H \rightarrow Register 00H = 00H

CLR @01H \rightarrow Register 01H = 02H, register 02H = 00H

In Register (R) addressing mode, the statement "CLR 00H" clears the destination register 00H value to 00H. In the second example, the statement "CLR @01H" uses Indirect Register (IR) addressing mode to clear the 02H register value to 00H.

${\color{red}\mathsf{COM}}$ — Complement

COM dst

Operation: $dst \leftarrow NOT dst$

The contents of the destination location are complemented (one's complement); all "1s" are changed to "0s", and vice-versa.

Flags: C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.S: Set if the result bit 7 is set; cleared otherwise.

V: Always reset to "0".

D: Unaffected.H: Unaffected.

Format:

		By	tes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
орс	dst	2	2	4	60	R
				4	61	IR

Examples: Given: R1 = 07H and register 07H = 0F1H:

COM R1 \rightarrow R1 = 0F8H

COM @R1 \rightarrow R1 = 07H, register 07H = 0EH

In the first example, destination working register R1 contains the value 07H (00000111B). The statement "COM R1" complements all the bits in R1: all logic ones are changed to logic zeros, and vice-versa, leaving the value 0F8H (11111000B).

In the second example, Indirect Register (IR) addressing mode is used to complement the value of destination register 07H (11110001B), leaving the new value 0EH (00001110B).

CP — Compare

CP dst,src

Operation: dst – src

The source operand is compared to (subtracted from) the destination operand, and the appropriate flags are set accordingly. The contents of both operands are unaffected by the comparison.

Companson

Flags: C: Set if a "borrow" occurred (src > dst); cleared otherwise.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result is negative; cleared otherwise.

V: Set if arithmetic overflow occurred; cleared otherwise.

D: Unaffected.H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr <u>dst</u>	Mode <u>src</u>
орс	dst src		2	4	A2	r	r
				6	А3	r	lr
орс	src	dst	3	6	A4	R	R
				6	A5	R	IR
орс	dst	src	3	6	A6	R	IM

Examples: 1. Given: R1 = 02H and R2 = 03H:

CP R1,R2 \rightarrow Set the C and S flags

Destination working register R1 contains the value 02H and source register R2 contains the value 03H. The statement "CP R1,R2" subtracts the R2 value (source/subtrahend) from the R1 value (destination/minuend). Because a "borrow" occurs and the difference is negative, C and S are "1".

2. Given: R1 = 05H and R2 = 0AH:

In this example, destination working register R1 contains the value 05H which is less than the contents of the source working register R2 (0AH). The statement "CP R1,R2" generates C = "1" and the JP instruction does not jump to the SKIP location. After the statement "LD R3,R1" executes, the value 06H remains in working register R3.



CPIJE — Compare, Increment, and Jump on Equal

CPIJE dst,src,RA

Operation: If dst - src = "0", $PC \leftarrow PC + RA$

 $lr \leftarrow lr + 1$

The source operand is compared to (subtracted from) the destination operand. If the result is "0", the relative address is added to the program counter and control passes to the statement whose address is now in the program counter. Otherwise, the instruction immediately following the CPIJE instruction is executed. In either case, the source pointer is incremented by one before the next instruction is executed.

Flags: No flags are affected.

Format:

				Byt	es	Cycles	Opcode (Hex)		
орс	src	dst	RA	3		12	C2	r	Ir

NOTE: Execution time is 18 cycles if the jump is taken or 16 cycles if it is not taken.

Example: Given: R1 = 02H, R2 = 03H, and register 03H = 02H:

CPIJE R1,@R2,SKIP \rightarrow R2 = 04H, PC jumps to SKIP location

In this example, working register R1 contains the value 02H, working register R2 the value 03H, and register 03 contains 02H. The statement "CPIJE R1,@R2,SKIP" compares the @R2 value 02H (00000010B) to 02H (00000010B). Because the result of the comparison is *equal*, the relative address is added to the PC and the PC then jumps to the memory location pointed to by SKIP. The source register (R2) is incremented by one, leaving a value of 04H. (Remember that the memory location must be within the allowed range of + 127 to - 128.)

CPIJNE — Compare, Increment, and Jump on Non-Equal

CPIJNE dst,src,RA

Operation: If dst - src = "0", $PC \leftarrow PC + RA$

 $lr \leftarrow lr + 1$

The source operand is compared to (subtracted from) the destination operand. If the result is not "0", the relative address is added to the program counter and control passes to the statement whose address is now in the program counter; otherwise the instruction following the CPIJNE instruction is executed. In either case the source pointer is incremented by one before the next instruction.

Flags: No flags are affected.

Format:

				Bytes	Cycles	Opcode	Addr	Mode
						(Hex)	<u>dst</u>	src
орс	src	dst	RA	3	12	D2	r	lr

NOTE: Execution time is 18 cycles if the jump is taken or 16 cycles if it is not taken.

Example: Given: R1 = 02H, R2 = 03H, and register 03H = 04H:

CPIJNE R1,@R2,SKIP \rightarrow R2 = 04H, PC jumps to SKIP location

Working register R1 contains the value 02H, working register R2 (the source pointer) the value 03H, and general register 03 the value 04H. The statement "CPIJNE R1,@R2,SKIP" subtracts 04H (00000100B) from 02H (00000010B). Because the result of the comparison is *non-equal*, the relative address is added to the PC and the PC then jumps to the memory location pointed to by SKIP. The source pointer register (R2) is also incremented by one, leaving a value of 04H. (Remember that the memory location must be within the allowed range of + 127 to - 128.)



DA — Decimal Adjust

DA dst

Operation: $dst \leftarrow DA dst$

The destination operand is adjusted to form two 4-bit BCD digits following an addition or subtraction operation. For addition (ADD, ADC) or subtraction (SUB, SBC), the following table indicates the operation performed. (The operation is undefined if the destination operand was not the result of a valid addition or subtraction of BCD digits):

Instruction	Carry Before DA	Bits 4–7 Value (Hex)	H Flag Before DA	Bits 0–3 Value (Hex)	Number Added to Byte	Carry After DA
	0	0–9	0	0–9	00	0
	0	0–8	0	A-F	06	0
	0	0–9	1	0–3	06	0
ADD	0	A-F	0	0–9	60	1
ADC	0	9–F	0	A-F	66	1
	0	A-F	1	0–3	66	1
	1	0–2	0	0–9	60	1
	1	0–2	0	A-F	66	1
	1	0–3	1	0–3	66	1
	0	0–9	0	0–9	00 = -00	0
SUB	0	0–8	1	6-F	FA = -06	0
SBC	1	7–F	0	0–9	A0 = -60	1
	1	6–F	1	6–F	9A = -66	1

Flags: C: Set if there was a carry from the most significant bit; cleared otherwise (see table).

Z: Set if result is "0"; cleared otherwise.

S: Set if result bit 7 is set; cleared otherwise.

V: Undefined.D: Unaffected.H: Unaffected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
орс	dst	2	4	40	R
			4	<i>4</i> 1	IR

DA — Decimal Adjust

DA (Continued)

Example: Given: Working register R0 contains the value 15 (BCD), working register R1 contains 27 (BCD), and address 27H contains 46 (BCD):

ADD R1,R0 ; $C \leftarrow "0"$, $H \leftarrow "0"$, Bits 4-7 = 3, bits 0-3 = C, $R1 \leftarrow 3CH$

DA R1 ; $R1 \leftarrow 3CH + 06$

If addition is performed using the BCD values 15 and 27, the result should be 42. The sum is incorrect, however, when the binary representations are added in the destination location using standard binary arithmetic:

The DA instruction adjusts this result so that the correct BCD representation is obtained:

$$0011 1100 + 0000 0110 = 42$$

Assuming the same values given above, the statements

SUB 27H,R0 ; $C \leftarrow "0", H \leftarrow "0", Bits 4-7 = 3, bits 0-3 = 1$

DA @R1 ; $@R1 \leftarrow 31-0$

leave the value 31 (BCD) in address 27H (@R1).

DEC — Decrement

DEC dst

Operation: $dst \leftarrow dst - 1$

The contents of the destination operand are decremented by one.

Flags: C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.S: Set if result is negative; cleared otherwise.

V: Set if arithmetic overflow occurred; cleared otherwise.

D: Unaffected.H: Unaffected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
орс	dst	2	4	00	R
			4	01	IR

Examples: Given: R1 = 03H and register 03H = 10H:

DEC R1 \rightarrow R1 = 02H

DEC @R1 \rightarrow Register 03H = 0FH

In the first example, if working register R1 contains the value 03H, the statement "DEC R1" decrements the hexadecimal value by one, leaving the value 02H. In the second example, the statement "DEC @R1" decrements the value 10H contained in the destination register 03H by one, leaving the value 0FH.

DECW — Decrement Word

DECW dst

Operation: $dst \leftarrow dst - 1$

The contents of the destination location (which must be an even address) and the operand following that location are treated as a single 16-bit value that is decremented by one.

Flags: C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result is negative; cleared otherwise.

V: Set if arithmetic overflow occurred; cleared otherwise.

D: Unaffected.H: Unaffected.

Format:

		Byte	s Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
орс	dst	2	8	80	RR
			8	81	IR

Examples: Given: R0 = 12H, R1 = 34H, R2 = 30H, register 30H = 0FH, and register 31H = 21H:

DECW RR0 \rightarrow R0 = 12H, R1 = 33H

DECW @R2 → Register 30H = 0FH, register 31H = 20H

In the first example, destination register R0 contains the value 12H and register R1 the value 34H. The statement "DECW RR0" addresses R0 and the following operand R1 as a 16-bit word and decrements the value of R1 by one, leaving the value 33H.

NOTE: A system malfunction may occur if you use a Zero flag (FLAGS.6) result together with a DECW

instruction. To avoid this problem, we recommend that you use DECW as shown in the following

example:

LOOP: DECW RR0

LD R2,R1 OR R2,R0 JR NZ,LOOP



${f DI}$ — Disable Interrupts

DI

Operation: SYM $(0) \leftarrow 0$

Bit zero of the system mode control register, SYM.0, is cleared to "0", globally disabling all interrupt processing. Interrupt requests will continue to set their respective interrupt pending bits,

but the CPU will not service them while interrupt processing is disabled.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
орс	1	4	8F

Example: Given: SYM = 01H:

DI

If the value of the SYM register is 01H, the statement "DI" leaves the new value 00H in the register and clears SYM.0 to "0", disabling interrupt processing.

Before changing IMR, interrupt pending and interrupt source control register, be sure DI state.

DIV — Divide (Unsigned)

DIV dst,src

Operation: dst ÷ src

 $\begin{array}{l} \mathsf{dst} \ (\mathsf{UPPER}) \leftarrow \mathsf{REMAINDER} \\ \mathsf{dst} \ (\mathsf{LOWER}) \ \leftarrow \ \mathsf{QUOTIENT} \end{array}$

The destination operand (16 bits) is divided by the source operand (8 bits). The quotient (8 bits) is stored in the lower half of the destination. The remainder (8 bits) is stored in the upper half of the destination. When the quotient is $\geq 2^8$, the numbers stored in the upper and lower halves of the destination for quotient and remainder are incorrect. Both operands are treated as unsigned integers.

Flags:

C: Set if the V flag is set and quotient is between 2^8 and $2^9 - 1$; cleared otherwise.

Z: Set if divisor or quotient = "0"; cleared otherwise.

S: Set if MSB of quotient = "1"; cleared otherwise.

V: Set if quotient is $\geq 2^8$ or if divisor = "0"; cleared otherwise.

D: Unaffected.H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr <u>dst</u>	Mode <u>src</u>
орс	src	dst	3	26/10	94	RR	R
				26/10	95	RR	IR
				26/10	96	RR	IM

NOTE: Execution takes 10 cycles if the divide-by-zero is attempted; otherwise it takes 26 cycles.

Examples: Given: R0 = 10H, R1 = 03H, R2 = 40H, register 40H = 80H:

DIV RR0,R2 \rightarrow R0 = 03H, R1 = 40H DIV RR0,@R2 \rightarrow R0 = 03H, R1 = 20H DIV RR0,#20H \rightarrow R0 = 03H, R1 = 80H

In the first example, destination working register pair RR0 contains the values 10H (R0) and 03H (R1), and register R2 contains the value 40H. The statement "DIV RR0,R2" divides the 16-bit RR0 value by the 8-bit value of the R2 (source) register. After the DIV instruction, R0 contains the value 03H and R1 contains 40H. The 8-bit remainder is stored in the upper half of the destination register RR0 (R0) and the quotient in the lower half (R1).



DJNZ — Decrement and Jump if Non-Zero

DJNZ r,dst

Operation: $r \leftarrow r - 1$

If r = 0, $PC \leftarrow PC + dst$

The working register being used as a counter is decremented. If the contents of the register are not logic zero after decrementing, the relative address is added to the program counter and control passes to the statement whose address is now in the PC. The range of the relative address is +127 to -128, and the original value of the PC is taken to be the address of the instruction byte following the DJNZ statement.

NOTE: In case of using DJNZ instruction, the working register being used as a counter should be set at

the one of location 0C0H to 0CFH with SRP, SRP0, or SRP1 instruction.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
r opc	dst	2	8 (jump taken)	rA	RA
			8 (no jump)	r = 0 to F	

Example: Given: R1 = 02H and LOOP is the label of a relative address:

SRP #0C0H DJNZ R1,LOOP

DJNZ is typically used to control a "loop" of instructions. In many cases, a label is used as the destination operand instead of a numeric relative address value. In the example, working register R1 contains the value 02H, and LOOP is the label for a relative address.

The statement "DJNZ R1, LOOP" decrements register R1 by one, leaving the value 01H. Because the contents of R1 after the decrement are non-zero, the jump is taken to the relative address specified by the LOOP label.

EI — Enable Interrupts

ΕI

Operation: SYM $(0) \leftarrow 1$

An EI instruction sets bit zero of the system mode register, SYM.0 to "1". This allows interrupts to be serviced as they occur (assuming they have highest priority). If an interrupt's pending bit was set while interrupt processing was disabled (by executing a DI instruction), it will be serviced

when you execute the EI instruction.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
орс	1	4	9F

Example: Given: SYM = 00H:

ΕI

If the SYM register contains the value 00H, that is, if interrupts are currently disabled, the statement "EI" sets the SYM register to 01H, enabling all interrupts. (SYM.0 is the enable bit for global interrupt processing.)

ENTER — Enter

ENTER

Operation: SP \leftarrow SP -2

 $\begin{array}{cccc} @\mathsf{SP} & \leftarrow & \mathsf{IP} \\ \mathsf{IP} & \leftarrow & \mathsf{PC} \\ \mathsf{PC} & \leftarrow & @\mathsf{IP} \\ \mathsf{IP} & \leftarrow & \mathsf{IP} + 2 \end{array}$

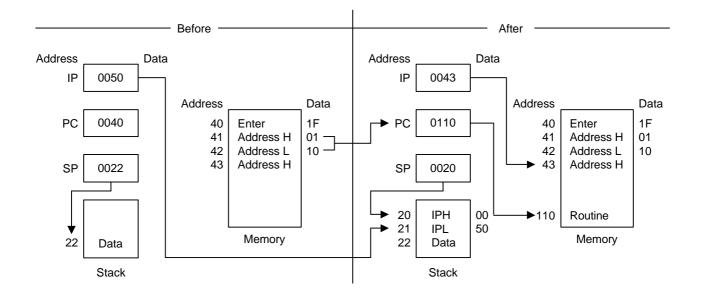
This instruction is useful when implementing threaded-code languages. The contents of the instruction pointer are pushed to the stack. The program counter (PC) value is then written to the instruction pointer. The program memory word that is pointed to by the instruction pointer is loaded into the PC, and the instruction pointer is incremented by two.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
орс	1	14	1F

Example: The diagram below shows one example of how to use an ENTER statement.



EXIT—Exit

EXIT

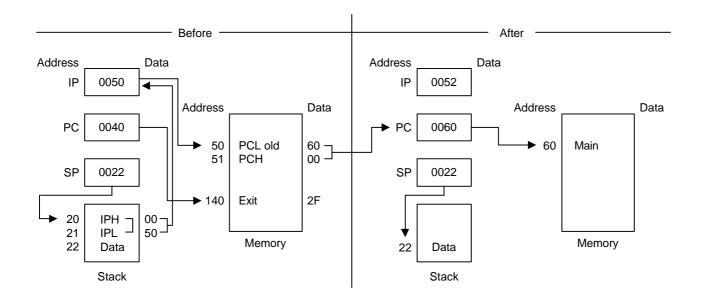
This instruction is useful when implementing threaded-code languages. The stack value is popped and loaded into the instruction pointer. The program memory word that is pointed to by the instruction pointer is then loaded into the program counter, and the instruction pointer is incremented by two.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
орс	1	14 (internal stack)	2F
		16 (internal stack)	

Example: The diagram below shows one example of how to use an EXIT statement.



IDLE — Idle Operation

IDLE

Operation:

The IDLE instruction stops the CPU clock while allowing system clock oscillation to continue. Idle

mode can be released by an interrupt request (IRQ) or an external reset operation.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	_	
орс	1	4	6F	_	_

Example: The instruction

IDLE NOP NOP NOP

stops the CPU clock but not the system clock.

INC — Increment

INC dst

Operation: $dst \leftarrow dst + 1$

The contents of the destination operand are incremented by one.

Flags: C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result is negative; cleared otherwise.

V: Set if arithmetic overflow occurred; cleared otherwise.

D: Unaffected.H: Unaffected.

Format:

		Byte	es Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
dst opc		1	4	rE	r
				r = 0 to F	
орс	dst	2	4	20	R
			4	21	IR

Examples: Given: R0 = 1BH, register 00H = 0CH, and register 1BH = 0FH:

INC R0
$$\rightarrow$$
 R0 = 1CH INC 00H \rightarrow Register 00H = 0DH INC @R0 \rightarrow R0 = 1BH, register 01H = 10H

In the first example, if destination working register R0 contains the value 1BH, the statement "INC R0" leaves the value 1CH in that same register.

The next example shows the effect an INC instruction has on register 00H, assuming that it contains the value 0CH.

In the third example, INC is used in Indirect Register (IR) addressing mode to increment the value of register 1BH from 0FH to 10H.



INCW — Increment Word

INCW dst

 $dst \leftarrow dst + 1$ Operation:

The contents of the destination (which must be an even address) and the byte following that

location are treated as a single 16-bit value that is incremented by one.

C: Unaffected. Flags:

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result is negative; cleared otherwise.

V: Set if arithmetic overflow occurred; cleared otherwise.

D: Unaffected. H: Unaffected.

Format:

		В	ytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
орс	dst		2	8	A0	RR
				8	A1	IR

Examples: Given: R0 = 1AH, R1 = 02H, register 02H = 0FH, and register 03H = 0FFH:

> **INCW** RR0 R0 = 1AH, R1 = 03H

INCW @R1 Register 02H = 10H, register 03H = 00H \rightarrow

In the first example, the working register pair RR0 contains the value 1AH in register R0 and 02H in register R1. The statement "INCW RR0" increments the 16-bit destination by one, leaving the value 03H in register R1. In the second example, the statement "INCW @R1" uses Indirect Register (IR) addressing mode to increment the contents of general register 03H from 0FFH to

00H and register 02H from 0FH to 10H.

NOTE: A system malfunction may occur if you use a Zero (Z) flag (FLAGS.6) result together with an INCW instruction. To avoid this problem, we recommend that you use INCW as shown in the

following example:

RR0 LOOP: **INCW**

LD R2,R1 OR R2,R0 JR NZ,LOOP

IRET — Interrupt Return

IRET (Normal) IRET (Fast)

 $SP \leftarrow SP + 1$ FLAGS \leftarrow FLAGS'

 $PC \leftarrow @SP \qquad FIS \leftarrow 0$

 $SP \leftarrow SP + 2$ $SYM(0) \leftarrow 1$

This instruction is used at the end of an interrupt service routine. It restores the flag register and the program counter. It also re-enables global interrupts. A "normal IRET" is executed only if the fast interrupt status bit (FIS, bit one of the FLAGS register, 0D5H) is cleared (= "0"). If a fast interrupt occurred, IRET clears the FIS bit that was set at the beginning of the service routine.

Flags: All flags are restored to their original settings (that is, the settings before the interrupt occurred).

Format:

IRET (Normal)	Bytes	Cycles	Opcode (Hex)
орс	1	10 (internal stack)	BF
		12 (internal stack)	
IRET (Fast)	Bytes	Cycles	Opcode (Hex)
орс	1	6	BF

Example:

In the figure below, the instruction pointer is initially loaded with 100H in the main program before interrupts are enabled. When an interrupt occurs, the program counter and instruction pointer are swapped. This causes the PC to jump to address 100H and the IP to keep the return address. The last instruction in the service routine normally is a jump to IRET at address FFH. This causes the instruction pointer to be loaded with 100H "again" and the program counter to jump back to the main program. Now, the next interrupt can occur and the IP is still correct at 100H.

0H		
FFH	IRET	
100H	Interrupt Service Routine	
	JP to FFH	
FFFFH		

NOTE:

In the fast interrupt example above, if the last instruction is not a jump to IRET, you must pay attention to the order of the last two instructions. The IRET cannot be immediately proceeded by a clearing of the interrupt status (as with a reset of the IPR register).



JP — Jump

JP cc,dst (Conditional)

JP dst (Unconditional)

Operation: If cc is true, $PC \leftarrow dst$

The conditional JUMP instruction transfers program control to the destination address if the condition specified by the condition code (cc) is true; otherwise, the instruction following the JP instruction is executed. The unconditional JP simply replaces the contents of the PC with the contents of the specified register pair. Control then passes to the statement addressed by the PC.

Flags: No flags are affected.

Format: (1)

(2)			Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
cc opc	dst		3	8	ccD	DA
					cc = 0 to F	
орс	dst		2	8	30	IRR

NOTES:

- 1. The 3-byte format is used for a conditional jump and the 2-byte format for an unconditional jump.
- 2. In the first byte of the three-byte instruction format (conditional jump), the condition code and the opcode are both four bits.

Examples: Given: The carry flag (C) = "1", register 00 = 01H, and register 01 = 20H:

JP C,LABEL_W
$$\rightarrow$$
 LABEL_W = 1000H, PC = 1000H JP @00H \rightarrow PC = 0120H

The first example shows a conditional JP. Assuming that the carry flag is set to "1", the statement

"JP C,LABEL_W" replaces the contents of the PC with the value 1000H and transfers control to that location. Had the carry flag not been set, control would then have passed to the statement immediately following the JP instruction.

The second example shows an unconditional JP. The statement "JP @00" replaces the contents of the PC with the contents of the register pair 00H and 01H, leaving the value 0120H.

JR — Jump Relative

JR cc,dst

Operation: If cc is true, $PC \leftarrow PC + dst$

If the condition specified by the condition code (cc) is true, the relative address is added to the program counter and control passes to the statement whose address is now in the program counter; otherwise, the instruction following the JR instruction is executed. (See list of condition codes).

The range of the relative address is +127, -128, and the original value of the program counter is taken to be the address of the first instruction byte following the JR statement.

Flags: No flags are affected.

Format:

(1)		_	Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
cc opc	dst		2	6	ссВ	RA
		_			cc = 0 to F	

NOTE: In the first byte of the two-byte instruction format, the condition code and the opcode are each four bits.

Example: Given: The carry flag = "1" and LABEL_X = 1FF7H:

JR C,LABEL X
$$\rightarrow$$
 PC = 1FF7H

If the carry flag is set (that is, if the condition code is true), the statement "JR C,LABEL_X" will pass control to the statement whose address is now in the PC. Otherwise, the program instruction following the JR would be executed.



LD — Load

LD dst,src

Operation: $dst \leftarrow src$

The contents of the source are loaded into the destination. The source's contents are unaffected.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr <u>dst</u>	Mode <u>src</u>
dst opc	src		2	4	rC	r	IM
				4	r8	r	R
src opc	dst		2	4	r9	R	r
					r = 0 to F		
орс	dst src		2	4	C7	r	lr
				4	D7	lr	r
орс	src	dst	3	6	E4	R	R
			I	6	E5	R	IR
орс	dst	src	3	6	E6	R	IM
			I	6	D6	IR	IM
орс	src	dst	3	6	F5	IR	R
			I				
орс	dst src	х	3	6	87	r	x [r]
	T		1				
орс	src dst	Х	3	6	97	x [r]	r

LD — Load

LD (Continued)

Examples: Given: R0 = 01H, R1 = 0AH, register 00H = 01H, register 01H = 20H,

register 02H = 02H, LOOP = 30H, and register 3AH = 0FFH:

LD R0,#10H \rightarrow R0 = 10H

LD R0,01H \rightarrow R0 = 20H, register 01H = 20H

LD 01H,R0 \rightarrow Register 01H = 01H, R0 = 01H

LD R1,@R0 \rightarrow R1 = 20H, R0 = 01H

LD @R0,R1 \rightarrow R0 = 01H, R1 = 0AH, register 01H = 0AH

LD 00H,01H \rightarrow Register 00H = 20H, register 01H = 20H

LD 02H,@00H \rightarrow Register 02H = 20H, register 00H = 01H

LD 00H,#0AH \rightarrow Register 00H = 0AH

LD @00H,#10H \rightarrow Register 00H = 01H, register 01H = 10H

LD @00H,02H \rightarrow Register 00H = 01H, register 01H = 02, register 02H = 02H

LD R0,#LOOP[R1] \rightarrow R0 = 0FFH, R1 = 0AH

LD #LOOP[R0],R1 \rightarrow Register 31H = 0AH, R0 = 01H, R1 = 0AH

LDB — Load Bit

LDB dst,src.b

LDB dst.b,src

Operation: $dst(0) \leftarrow src(b)$

or

 $dst(b) \leftarrow src(0)$

The specified bit of the source is loaded into bit zero (LSB) of the destination, or bit zero of the source is loaded into the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

Flags: No flags are affected.

Format:

			Byt	es Cycle	es Opcod (Hex)		r Mode <u>src</u>
орс	dst b 0	src	3	6	47	r0	Rb
орс	src b 1	dst	3	6	47	Rb	r0

NOTE: In the second byte of the instruction formats, the destination (or source) address is four bits, the bit address 'b' is three bits, and the LSB address value is one bit in length.

Examples: Given: R0 = 06H and general register 00H = 05H:

LDB R0,00H.2 \rightarrow R0 = 07H, register 00H = 05H LDB 00H.0,R0 \rightarrow R0 = 06H, register 00H = 04H

In the first example, destination working register R0 contains the value 06H and the source general register 00H the value 05H. The statement "LD R0,00H.2" loads the bit two value of the 00H register into bit zero of the R0 register, leaving the value 07H in register R0.

In the second example, 00H is the destination register. The statement "LD 00H.0,R0" loads bit zero of register R0 to the specified bit (bit zero) of the destination register, leaving 04H in general register 00H.

LDC/LDE — Load Memory

LDC/LDE dst,src

Operation: $dst \leftarrow src$

This instruction loads a byte from program or data memory into a working register or vice-versa. The source values are unaffected. LDC refers to program memory and LDE to data memory. The assembler makes 'Irr' or 'rr' values an even number for program memory and odd an odd number for data memory.

Flags: No flags are affected.

Format:

					Bytes	Cycles	Opcode (Hex)	Addr <u>dst</u>	Mode <u>src</u>
1.	орс	dst src			2	10	C3	r	Irr
2.	орс	src dst			2	10	D3	Irr	r
3.	орс	dst src	XS]	3	12	E7	r	XS [rr]
4.	орс	src dst	XS]	3	12	F7	XS [rr]	r
5.	орс	dst src	XLL	XLH	4	14	A7	r	XL [rr]
6.	орс	src dst	XLL	XL _H	4	14	В7	XL [rr]	r
7.	орс	dst 0000	DAL	DA _H	4	14	A7	r	DA
8.	opc	src 0000	DAL	DA _H	4	14	В7	DA	r
9.	орс	dst 0001	DAL	DA _H	4	14	A7	r	DA
10.	opc	src 0001	DAL	DA _H	4	14	В7	DA	r

NOTES:

- 1. The source (src) or working register pair [rr] for formats 5 and 6 cannot use register pair 0–1.
- 2. For formats 3 and 4, the destination address 'XS [rr]' and the source address 'XS [rr]' are each one byte.
- 3. For formats 5 and 6, the destination address 'XL [rr] and the source address 'XL [rr]' are each two bytes.
- 4. The DA and r source values for formats 7 and 8 are used to address program memory; the second set of values, used in formats 9 and 10, are used to address data memory.



LDC/LDE — Load Memory

LDC/LDE (Continued)

Examples: Given: R0 = 11H, R1 = 34H, R2 = 01H, R3 = 04H; Program memory locations

0103H = 4FH, 0104H = 1A, 0105H = 6DH, and 1104H = 88H. External data memory

locations 0103H = 5FH, 0104H = 2AH, 0105H = 7DH, and 1104H = 98H:

LDC R0,@RR2 ; R0 ← contents of program memory location 0104H

; R0 = 1AH, R2 = 01H, R3 = 04H

LDE R0,@RR2 ; R0 ← contents of external data memory location 0104H

R0 = 2AH, R2 = 01H, R3 = 04H

LDC (note) @RR2,R0 ; 11H (contents of R0) is loaded into program memory

; location 0104H (RR2),

; working registers R0, R2, R3 → no change

LDE @RR2,R0 ; 11H (contents of R0) is loaded into external data memory

; location 0104H (RR2),

working registers R0, R2, R3 \rightarrow no change

LDC R0,#01H[RR2] ; R0 ← contents of program memory location 0105H

(01H + RR2),

R0 = 6DH, R2 = 01H, R3 = 04H

LDE R0,#01H[RR2]; R0 ← contents of external data memory location 0105H

(01H + RR2), R0 = 7DH, R2 = 01H, R3 = 04H

LDC (note) #01H[RR2],R0 ; 11H (contents of R0) is loaded into program memory location

0105H (01H + 0104H)

LDE #01H[RR2],R0 ; 11H (contents of R0) is loaded into external data memory

location 0105H (01H + 0104H)

LDC R0,#1000H[RR2]; R0 ← contents of program memory location 1104H

(1000H + 0104H), R0 = 88H, R2 = 01H, R3 = 04H

LDE R0,#1000H[RR2]; R0 ← contents of external data memory location 1104H

(1000H + 0104H), R0 = 98H, R2 = 01H, R3 = 04H

LDC R0,1104H ; R0 ← contents of program memory location 1104H, R0 = 88H

LDE R0,1104H ; R0 ← contents of external data memory location 1104H,

R0 = 98H

LDC (note) 1105H,R0 ; 11H (contents of R0) is loaded into program memory location

; 1105H, $(1105H) \leftarrow 11H$

LDE 1105H,R0 ; 11H (contents of R0) is loaded into external data memory

location 1105H, (1105H) \leftarrow 11H

NOTE: These instructions are not supported by masked ROM type devices.



LDCD/LDED — Load Memory and Decrement

LDCD/LDED dst,src

Operation: $dst \leftarrow src$

 $rr \leftarrow rr - 1$

These instructions are used for user stacks or block transfers of data from program or data memory to the register file. The address of the memory location is specified by a working register pair. The contents of the source location are loaded into the destination location. The memory address is then decremented. The contents of the source are unaffected.

LDCD references program memory and LDED references external data memory. The assembler makes 'Irr' an even number for program memory and an odd number for data memory.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode	Addr Mode	
				(Hex)	<u>dst</u>	src
орс	dst src	2	10	E2	r	Irr

Examples: Given: R6 = 10H, R7 = 33H, R8 = 12H, program memory location 1033H = 0CDH, and

external data memory location 1033H = 0DDH:

LDCD R8,@RR6 ; 0CDH (contents of program memory location 1033H) is loaded

; into R8 and RR6 is decremented by one

; R8 = 0CDH, R6 = 10H, R7 = 32H (RR6 \leftarrow RR6 - 1)

LDED R8,@RR6; 0DDH (contents of data memory location 1033H) is loaded

; into R8 and RR6 is decremented by one (RR6 \leftarrow RR6 – 1)

R8 = 0DDH, R6 = 10H, R7 = 32H

LDCI/LDEI — Load Memory and Increment

LDCI/LDEI dst,src

Operation: $dst \leftarrow src$

 $rr \leftarrow rr + 1$

These instructions are used for user stacks or block transfers of data from program or data memory to the register file. The address of the memory location is specified by a working register pair. The contents of the source location are loaded into the destination location. The memory address is then incremented automatically. The contents of the source are unaffected.

LDCI refers to program memory and LDEI refers to external data memory. The assembler makes 'Irr' even for program memory and odd for data memory.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode	Addr Mode	
				(Hex)	<u>dst</u>	<u>src</u>
орс	dst src	2	10	E3	r	Irr

Examples: Given: R6 = 10H, R7 = 33H, R8 = 12H, program memory locations 1033H = 0CDH and 1034H = 0C5H; external data memory locations 1033H = 0DDH and 1034H = 0D5H:

LDCI R8,@RR6 ; 0CDH (contents of program memory location 1033H) is loaded

into R8 and RR6 is incremented by one (RR6 \leftarrow RR6 + 1)

; R8 = 0CDH, R6 = 10H, R7 = 34H

LDEI R8,@RR6; 0DDH (contents of data memory location 1033H) is loaded

into R8 and RR6 is incremented by one (RR6 \leftarrow RR6 + 1)

; R8 = 0DDH, R6 = 10H, R7 = 34H

LDCPD/LDEPD — Load Memory with Pre-Decrement

LDCPD/

LDEPD dst,src

Operation: $rr \leftarrow rr - 1$

 $dst \leftarrow src$

These instructions are used for block transfers of data from program or data memory from the register file. The address of the memory location is specified by a working register pair and is first decremented. The contents of the source location are then loaded into the destination location. The contents of the source are unaffected.

LDCPD refers to program memory and LDEPD refers to external data memory. The assembler makes 'Irr' an even number for program memory and an odd number for external data memory.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode	Addr Mode	
				(Hex)	<u>dst</u>	<u>src</u>
орс	src dst	2	14	F2	Irr	r

Examples: Given: R0 = 77H, R6 = 30H, and R7 = 00H:

LDCPD @RR6,R0 ; $(RR6 \leftarrow RR6 - 1)$

77H (contents of R0) is loaded into program memory location

2FFFH (3000H - 1H)

R0 = 77H, R6 = 2FH, R7 = 0FFH

LDEPD @RR6,R0 ; $(RR6 \leftarrow RR6 - 1)$

; 77H (contents of R0) is loaded into external data memory

location 2FFFH (3000H – 1H)

; R0 = 77H, R6 = 2FH, R7 = 0FFH



LDCPI/LDEPI — Load Memory with Pre-Increment

LDCPI/

LDEPI dst,src

Operation: $rr \leftarrow rr + 1$

 $dst \leftarrow src$

These instructions are used for block transfers of data from program or data memory from the register file. The address of the memory location is specified by a working register pair and is first incremented. The contents of the source location are loaded into the destination location. The contents of the source are unaffected.

LDCPI refers to program memory and LDEPI refers to external data memory. The assembler makes 'Irr' an even number for program memory and an odd number for data memory.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode		
				(Hex)	<u>dst</u>	<u>src</u>
орс	src dst	2	14	F3	Irr	r

Examples: Given: R0 = 7FH, R6 = 21H, and R7 = 0FFH:

LDCPI @RR6,R0 ; $(RR6 \leftarrow RR6 + 1)$

; 7FH (contents of R0) is loaded into program memory

; location 2200H (21FFH + 1H) ; R0 = 7FH, R6 = 22H, R7 = 00H

LDEPI @RR6,R0 ; $(RR6 \leftarrow RR6 + 1)$

7FH (contents of R0) is loaded into external data memory

; location 2200H (21FFH + 1H) ; R0 = 7FH, R6 = 22H, R7 = 00H

LDW — Load Word

LDW dst,src

Operation: $dst \leftarrow src$

The contents of the source (a word) are loaded into the destination. The contents of the source

are unaffected.

Flags: No flags are affected.

Format:

				Bytes	Cycles	Opcode (Hex)	Addr <u>dst</u>	Mode <u>src</u>
орс	src	dst		3	8	C4	RR	RR
					8	C5	RR	IR
орс	dst	s	c	4	8	C6	RR	IML

Examples: Given: R4 = 06H, R5 = 1CH, R6 = 05H, R7 = 02H, register 00H = 1AH, register 01H = 02H, register 02H = 03H, and register 03H = 0FH:

LDW RR6,RR4
$$\rightarrow$$
 R6 = 06H, R7 = 1CH, R4 = 06H, R5 = 1CH

LDW 00H,02H \rightarrow Register 00H = 03H, register 01H = 0FH, register 02H = 03H, register 03H = 0FH

LDW RR2,@R7 \rightarrow R2 = 03H, R3 = 0FH,

LDW 04H,@01H \rightarrow Register 04H = 03H, register 05H = 0FH

LDW RR6,#1234H \rightarrow R6 = 12H, R7 = 34H

LDW 02H,#0FEDH \rightarrow Register 02H = 0FH, register 03H = 0EDH

In the second example, please note that the statement "LDW 00H,02H" loads the contents of the source word 02H, 03H into the destination word 00H, 01H. This leaves the value 03H in general register 00H and the value 0FH in register 01H.

The other examples show how to use the LDW instruction with various addressing modes and formats.

MULT — Multiply (Unsigned)

MULT dst,src

Operation: $dst \leftarrow dst \times src$

The 8-bit destination operand (even register of the register pair) is multiplied by the source operand (8 bits) and the product (16 bits) is stored in the register pair specified by the destination

address. Both operands are treated as unsigned integers.

Flags: C: Set if result is > 255; cleared otherwise.

Z: Set if the result is "0"; cleared otherwise.

S: Set if MSB of the result is a "1"; cleared otherwise.

V: Cleared.D: Unaffected.H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr <u>dst</u>	Mode <u>src</u>
opc	src	dst	3	22	84	RR	R
				22	85	RR	IR
				22	86	RR	IM

Examples: Given: Register 00H = 20H, register 01H = 03H, register 02H = 09H, register 03H = 06H:

MULT 00H, 02H \rightarrow Register 00H = 01H, register 01H = 20H, register 02H = 09H MULT 00H, @01H \rightarrow Register 00H = 00H, register 01H = 0C0H MULT 00H, #30H \rightarrow Register 00H = 06H, register 01H = 00H

In the first example, the statement "MULT 00H,02H" multiplies the 8-bit destination operand (in the register 00H of the register pair 00H, 01H) by the source register 02H operand (09H). The 16-bit product, 0120H, is stored in the register pair 00H, 01H.

NEXT — Next

NEXT

Operation: $PC \leftarrow @ IP$

 $IP \leftarrow IP + 2$

The NEXT instruction is useful when implementing threaded-code languages. The program memory word that is pointed to by the instruction pointer is loaded into the program counter. The

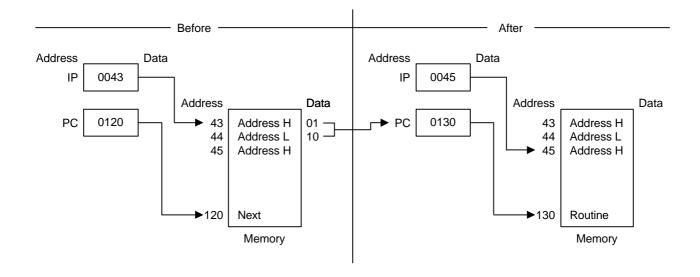
instruction pointer is then incremented by two.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
орс	1	10	0F

Example: The following diagram shows one example of how to use the NEXT instruction.



NOP — No Operation

NOP

Operation: No action is performed when the CPU executes this instruction. Typically, one or more NOPs are

executed in sequence in order to effect a timing delay of variable duration.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
орс	1	4	FF

Example: When the instruction

NOP

is encountered in a program, no operation occurs. Instead, there is a delay in instruction execution time.

OR — Logical OR

OR dst,src

Operation: $dst \leftarrow dst \ OR \ src$

The source operand is logically ORed with the destination operand and the result is stored in the destination. The contents of the source are unaffected. The OR operation results in a "1" being stored whenever either of the corresponding bits in the two operands is a "1"; otherwise a "0" is stored.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result bit 7 is set; cleared otherwise.

V: Always cleared to "0".

D: Unaffected.H: Unaffected.

C: Unaffected.

Format:

Flags:

					Bytes	Cycles	Opcode (Hex)	Addr <u>dst</u>	Mode <u>src</u>
0	рс	dst src			2	4	42	r	r
						6	43	r	lr
				1					
0	рс	src	dst		3	6	44	R	R
						6	45	R	IR
				ī					
0	рс	dst	src		3	6	46	R	IM

Examples:

Given: R0 = 15H, R1 = 2AH, R2 = 01H, register 00H = 08H, register 01H = 37H, and register 08H = 8AH:

OR R0,R1
$$\rightarrow$$
 R0 = 3FH, R1 = 2AH
OR R0,@R2 \rightarrow R0 = 37H, R2 = 01H, register 01H = 37H
OR 00H,01H \rightarrow Register 00H = 3FH, register 01H = 37H
OR 01H,@00H \rightarrow Register 00H = 08H, register 01H = 0BFH
OR 00H,#02H \rightarrow Register 00H = 0AH

In the first example, if working register R0 contains the value 15H and register R1 the value 2AH, the statement "OR R0,R1" logical-ORs the R0 and R1 register contents and stores the result (3FH) in destination register R0.

The other examples show the use of the logical OR instruction with the various addressing modes and formats.



POP — Pop From Stack

POP dst

Operation: $dst \leftarrow @SP$

 $SP \leftarrow SP + 1$

The contents of the location addressed by the stack pointer are loaded into the destination. The

stack pointer is then incremented by one.

Flags: No flags affected.

Format:

_			Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
	орс	dst	2	8	50	R
				8	51	IR

Examples: Given: Register 00H = 01H, register 01H = 1BH, SPH (0D8H) = 00H, SPL (0D9H) = 0FBH, and stack register 0FBH = 55H:

POP 00H \rightarrow Register 00H = 55H, SP = 00FCH POP @00H \rightarrow Register 00H = 01H, register 01H = 55H, SP = 00FCH

In the first example, general register 00H contains the value 01H. The statement "POP 00H" loads the contents of location 00FBH (55H) into destination register 00H and then increments the stack pointer by one. Register 00H then contains the value 55H and the SP points to location 00FCH.

POPUD — Pop User Stack (Decrementing)

POPUD dst,src

Operation: $dst \leftarrow src$

 $IR \leftarrow IR - 1$

This instruction is used for user-defined stacks in the register file. The contents of the register file location addressed by the user stack pointer are loaded into the destination. The user stack

pointer is then decremented.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode	Addr	Mode
					(Hex)	<u>dst</u>	src
opc	src	dst	3	8	92	R	IR

Example: Given: Register 00H = 42H (user stack pointer register), register 42H = 6FH, and register 02H = 70H:

POPUD 02H,@00H → Register 00H = 41H, register 02H = 6FH, register 42H = 6FH

If general register 00H contains the value 42H and register 42H the value 6FH, the statement "POPUD 02H,@00H" loads the contents of register 42H into the destination register 02H. The user stack pointer is then decremented by one, leaving the value 41H.

POPUI — Pop User Stack (Incrementing)

POPUI dst,src

Operation: $dst \leftarrow src$

 $IR \leftarrow IR + 1$

The POPUI instruction is used for user-defined stacks in the register file. The contents of the register file location addressed by the user stack pointer are loaded into the destination. The user

stack pointer is then incremented.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode	Addr	Mode
					(Hex)	<u>dst</u>	src
орс	src	dst	3	8	93	R	IR

Example: Given: Register 00H = 01H and register 01H = 70H:

POPUI 02H,@00H → Register 00H = 02H, register 01H = 70H, register 02H = 70H

If general register 00H contains the value 01H and register 01H the value 70H, the statement "POPUI 02H,@00H" loads the value 70H into the destination general register 02H. The user stack pointer (register 00H) is then incremented by one, changing its value from 01H to 02H.

PUSH — Push To Stack

PUSH src

Operation: $SP \leftarrow SP - 1$

 $@SP \leftarrow src$

A PUSH instruction decrements the stack pointer value and loads the contents of the source (src) into the location addressed by the decremented stack pointer. The operation then adds the new value to the top of the stack.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
орс	src	2	8 (internal clock)	70	R
			8 (external clock)		
			8 (internal clock)		
			8 (external clock)	71	IR

Examples: Given: Register 40H = 4FH, register 4FH = 0AAH, SPH = 00H, and SPL = 00H:

PUSH 40H \rightarrow Register 40H = 4FH, stack register 0FFH = 4FH, SPH = 0FFH, SPL = 0FFH

PUSH @40H \rightarrow Register 40H = 4FH, register 4FH = 0AAH, stack register 0FFH = 0AAH, SPH = 0FFH, SPL = 0FFH

In the first example, if the stack pointer contains the value 0000H, and general register 40H the value 4FH, the statement "PUSH 40H" decrements the stack pointer from 0000 to 0FFFFH. It then loads the contents of register 40H into location 0FFFFH and adds this new value to the top of the stack.



PUSHUD — Push User Stack (Decrementing)

PUSHUD dst,src

Operation: $IR \leftarrow IR - 1$

 $dst \,\leftarrow\, src$

This instruction is used to address user-defined stacks in the register file. PUSHUD decrements the user stack pointer and loads the contents of the source into the register addressed by the

decremented stack pointer.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode	Addr	Mode
					(Hex)	<u>dst</u>	<u>src</u>
орс	dst	src	3	8	82	IR	R

Example: Given: Register 00H = 03H, register 01H = 05H, and register 02H = 1AH:

PUSHUD @00H,01H → Register 00H = 02H, register 01H = 05H, register 02H = 05H

If the user stack pointer (register 00H, for example) contains the value 03H, the statement "PUSHUD @00H,01H" decrements the user stack pointer by one, leaving the value 02H. The 01H register value, 05H, is then loaded into the register addressed by the decremented user stack pointer.

PUSHUI — Push User Stack (Incrementing)

PUSHUI dst,src

Operation: $IR \leftarrow IR + 1$

 $dst \, \leftarrow \, src$

This instruction is used for user-defined stacks in the register file. PUSHUI increments the user stack pointer and then loads the contents of the source into the register location addressed by

the incremented user stack pointer.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode	Addr	Mode
					(Hex)	<u>dst</u>	src
орс	dst	src	3	8	83	IR	R

Example: Given: Register 00H = 03H, register 01H = 05H, and register 04H = 2AH:

PUSHUI @00H,01H → Register 00H = 04H, register 01H = 05H, register 04H = 05H

If the user stack pointer (register 00H, for example) contains the value 03H, the statement "PUSHUI @00H,01H" increments the user stack pointer by one, leaving the value 04H. The 01H register value, 05H, is then loaded into the location addressed by the incremented user stack pointer.



RCF — Reset Carry Flag

RCF RCF

Operation: $C \leftarrow 0$

The carry flag is cleared to logic zero, regardless of its previous value.

Flags: C: Cleared to "0".

No other flags are affected.

Format:

Bytes Cycles Opcode (Hex)

opc 1 4 CF

Example: Given: C = "1" or "0":

The instruction RCF clears the carry flag (C) to logic zero.

RET — Return

RET

Operation: $PC \leftarrow @SP$

 $SP \leftarrow SP + 2$

The RET instruction is normally used to return to the previously executing procedure at the end of a procedure entered by a CALL instruction. The contents of the location addressed by the stack pointer are popped into the program counter. The next statement that is executed is the one that is addressed by the new program counter value.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
орс	1	8 (internal stack)	AF
		10 (internal stack)	

Example: Given: SP = 00FCH, (SP) = 101AH, and PC = 1234:

RET \rightarrow PC = 101AH, SP = 00FEH

The statement "RET" pops the contents of stack pointer location 00FCH (10H) into the high byte of the program counter. The stack pointer then pops the value in location 00FEH (1AH) into the PC's low byte and the instruction at location 101AH is executed. The stack pointer now points to memory location 00FEH.

RL — Rotate Left

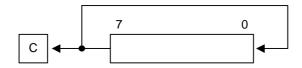
RL dst

Operation: $C \leftarrow dst(7)$

 $dst(0) \leftarrow dst(7)$

 $dst(n + 1) \leftarrow dst(n), n = 0-6$

The contents of the destination operand are rotated left one bit position. The initial value of bit 7 is moved to the bit zero (LSB) position and also replaces the carry flag.



Flags: C: Set if the bit rotated from the most significant bit position (bit 7) was "1".

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result bit 7 is set; cleared otherwise.

V: Set if arithmetic overflow occurred; cleared otherwise.

D: Unaffected.H: Unaffected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
орс	dst	2	4	90	R
			4	91	IR

Examples: Given: Register 00H = 0AAH, register 01H = 02H and register 02H = 17H:

RL 00H \rightarrow Register 00H = 55H, C = "1"

RL @01H \rightarrow Register 01H = 02H, register 02H = 2EH, C = "0"

In the first example, if general register 00H contains the value 0AAH (10101010B), the statement "RL 00H" rotates the 0AAH value left one bit position, leaving the new value 55H (01010101B) and setting the carry and overflow flags.

RLC — Rotate Left Through Carry

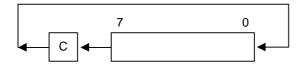
RLC dst

Operation: $dst(0) \leftarrow C$

 $C \leftarrow dst(7)$

 $dst(n + 1) \leftarrow dst(n), n = 0-6$

The contents of the destination operand with the carry flag are rotated left one bit position. The initial value of bit 7 replaces the carry flag (C); the initial value of the carry flag replaces bit zero.



Flags: C: Set if the bit rotated from the most significant bit position (bit 7) was "1".

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result bit 7 is set; cleared otherwise.

V: Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.

D: Unaffected.

H: Unaffected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
орс	dst	2	4	10	R
			4	11	IR

Examples: Given: Register 00H = 0AAH, register 01H = 02H, and register 02H = 17H, C = "0":

RLC 00H \rightarrow Register 00H = 54H, C = "1"

RLC @01H \rightarrow Register 01H = 02H, register 02H = 2EH, C = "0"

In the first example, if general register 00H has the value 0AAH (10101010B), the statement "RLC 00H" rotates 0AAH one bit position to the left. The initial value of bit 7 sets the carry flag and the initial value of the C flag replaces bit zero of register 00H, leaving the value 55H (01010101B). The MSB of register 00H resets the carry flag to "1" and sets the overflow flag.

RR — Rotate Right

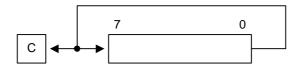
RR dst

Operation: $C \leftarrow dst(0)$

 $dst(7) \leftarrow dst(0)$

 $dst(n) \leftarrow dst(n + 1), n = 0-6$

The contents of the destination operand are rotated right one bit position. The initial value of bit zero (LSB) is moved to bit 7 (MSB) and also replaces the carry flag (C).



Flags: C: Set if the bit rotated from the least significant bit position (bit zero) was "1".

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result bit 7 is set; cleared otherwise.

V: Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.

D: Unaffected. **H:** Unaffected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
орс	dst	2	4	E0	R
			4	F1	IR

Examples: Given: Register 00H = 31H, register 01H = 02H, and register 02H = 17H:

RR 00H \rightarrow Register 00H = 98H, C = "1"

RR @01H \rightarrow Register 01H = 02H, register 02H = 8BH, C = "1"

In the first example, if general register 00H contains the value 31H (00110001B), the statement "RR 00H" rotates this value one bit position to the right. The initial value of bit zero is moved to bit 7, leaving the new value 98H (10011000B) in the destination register. The initial bit zero also resets the C flag to "1" and the sign flag and overflow flag are also set to "1".

RRC — Rotate Right Through Carry

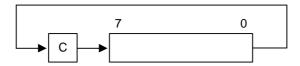
RRC dst

Operation: $dst(7) \leftarrow C$

 $C \leftarrow dst(0)$

 $dst(n) \leftarrow dst(n + 1), n = 0-6$

The contents of the destination operand and the carry flag are rotated right one bit position. The initial value of bit zero (LSB) replaces the carry flag; the initial value of the carry flag replaces bit 7 (MSB).



Flags:

- C: Set if the bit rotated from the least significant bit position (bit zero) was "1".
- Z: Set if the result is "0" cleared otherwise.
- **S:** Set if the result bit 7 is set; cleared otherwise.
- V: Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.
- D: Unaffected.
- H: Unaffected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
орс	dst	2	4	C0	R
			4	C1	IR

Examples: Given: Register 00H = 55H, register 01H = 02H, register 02H = 17H, and C = "0":

RRC 00H \rightarrow Register 00H = 2AH, C = "1"

RRC @01H \rightarrow Register 01H = 02H, register 02H = 0BH, C = "1"

In the first example, if general register 00H contains the value 55H (01010101B), the statement "RRC 00H" rotates this value one bit position to the right. The initial value of bit zero ("1") replaces the carry flag and the initial value of the C flag ("1") replaces bit 7. This leaves the new value 2AH (00101010B) in destination register 00H. The sign flag and overflow flag are both cleared to "0".

SB0 — Select Bank 0

SB0

Operation: BANK \leftarrow 0

The SB0 instruction clears the bank address flag in the FLAGS register (FLAGS.0) to logic zero,

selecting bank 0 register addressing in the set 1 area of the register file.

Flags: No flags are affected.

Format:

 Bytes
 Cycles
 Opcode (Hex)

 opc
 1
 4
 4F

Example: The statement

SB0

clears FLAGS.0 to "0", selecting bank 0 register addressing.

SB1 — Select Bank 1

SB1

Operation: BANK \leftarrow 1

The SB1 instruction sets the bank address flag in the FLAGS register (FLAGS.0) to logic one,

selecting bank 1 register addressing in the set 1 area of the register file. (Bank 1 is not

implemented in some S3C8-series microcontrollers.)

Flags: No flags are affected.

Format:

Bytes Cycles Opcode (Hex)

opc 1 4 5F

Example: The statement

SB1

sets FLAGS.0 to "1", selecting bank 1 register addressing, if implemented.

SBC — Subtract With Carry

SBC dst,src

Operation: $dst \leftarrow dst - src - c$

The source operand, along with the current value of the carry flag, is subtracted from the destination operand and the result is stored in the destination. The contents of the source are unaffected. Subtraction is performed by adding the two's-complement of the source operand to the destination operand. In multiple precision arithmetic, this instruction permits the carry ("borrow") from the subtraction of the low-order operands to be subtracted from the subtraction of high-order operands.

Flags: C: Set if a borrow occurred (src > dst); cleared otherwise.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result is negative; cleared otherwise.

V: Set if arithmetic overflow occurred, that is, if the operands were of opposite sign and the sign of the result is the same as the sign of the source; cleared otherwise.

D: Always set to "1".

H: Cleared if there is a carry from the most significant bit of the low-order four bits of the result; set otherwise, indicating a "borrow".

Format:

			Bytes	s Cycles	Opcode (Hex)	Addr <u>dst</u>	Mode <u>src</u>
орс	dst src		2	4	32	r	r
				6	33	r	lr
орс	src	dst	3	6	34	R	R
				6	35	R	IR
орс	dst	src	3	6	36	R	IM

Examples:

Given: R1 = 10H, R2 = 03H, C = "1", register 01H = 20H, register 02H = 03H, and register 03H = 0AH:

SBC R1,R2
$$\rightarrow$$
 R1 = 0CH, R2 = 03H
SBC R1,@R2 \rightarrow R1 = 05H, R2 = 03H, register 03H = 0AH
SBC 01H,02H \rightarrow Register 01H = 1CH, register 02H = 03H
SBC 01H,@02H \rightarrow Register 01H = 15H,register 02H = 03H, register 03H = 0AH
SBC 01H,#8AH \rightarrow Register 01H = 95H; C, S, and V = "1"

In the first example, if working register R1 contains the value 10H and register R2 the value 03H, the statement "SBC R1,R2" subtracts the source value (03H) and the C flag value ("1") from the destination (10H) and then stores the result (0CH) in register R1.

SCF — Set Carry Flag

SCF

Operation: $C \leftarrow 1$

The carry flag (C) is set to logic one, regardless of its previous value.

Flags: C: Set to "1".

No other flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
орс	1	4	DF

Example: The statement

SCF

sets the carry flag to logic one.

SRA — Shift Right Arithmetic

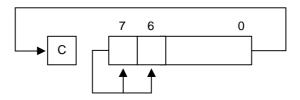
SRA dst

Operation: $dst(7) \leftarrow dst(7)$

 $C \leftarrow dst(0)$

 $dst(n) \leftarrow dst(n + 1), n = 0-6$

An arithmetic shift-right of one bit position is performed on the destination operand. Bit zero (the LSB) replaces the carry flag. The value of bit 7 (the sign bit) is unchanged and is shifted into bit position 6.



Flags: C: Set if the bit shifted from the LSB position (bit zero) was "1".

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result is negative; cleared otherwise.

V: Always cleared to "0".

D: Unaffected.
H: Unaffected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
орс	dst	2	4	D0	R
			4	D1	IR

Examples: Given: Register 00H = 9AH, register 02H = 03H, register 03H = 0BCH, and C = "1":

SRA 00H \rightarrow Register 00H = 0CD, C = "0"

SRA @02H \rightarrow Register 02H = 03H, register 03H = 0DEH, C = "0"

In the first example, if general register 00H contains the value 9AH (10011010B), the statement "SRA 00H" shifts the bit values in register 00H right one bit position. Bit zero ("0") clears the C flag and bit 7 ("1") is then shifted into the bit 6 position (bit 7 remains unchanged). This leaves the value 0CDH (11001101B) in destination register 00H.

SRP/SRP0/SRP1 — Set Register Pointer

SRP src

SRP0 src

SRP1 src

Operation: If src(1) = 1 and src(0) = 0 then: RP0 (3–7) \leftarrow src(3–7)

If $\operatorname{src}(1) = 0$ and $\operatorname{src}(0) = 1$ then: RP1 (3–7) $\leftarrow \operatorname{src}(3-7)$ If $\operatorname{src}(1) = 0$ and $\operatorname{src}(0) = 0$ then: RP0 (4–7) $\leftarrow \operatorname{src}(4-7)$,

RP0 (3) \leftarrow 0

RP1 $(4-7) \leftarrow src (4-7)$,

RP1 (3) \leftarrow 1

The source data bits one and zero (LSB) determine whether to write one or both of the register pointers, RP0 and RP1. Bits 3–7 of the selected register pointer are written unless both register pointers are selected. RP0.3 is then cleared to logic zero and RP1.3 is set to logic one.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>src</u>
орс	src	2	4	31	IM

Examples: The statement

SRP #40H

sets register pointer 0 (RP0) at location 0D6H to 40H and register pointer 1 (RP1) at location 0D7H to 48H.

The statement "SRP0 #50H" sets RP0 to 50H, and the statement "SRP1 #68H" sets RP1 to 68H.



STOP — Stop Operation

STOP

Operation:

The STOP instruction stops the both the CPU clock and system clock and causes the microcontroller to enter Stop mode. During Stop mode, the contents of on-chip CPU registers, peripheral registers, and I/O port control and data registers are retained. Stop mode can be released by an external reset operation or by external interrupts. For the reset operation, the RESET pin must be held to Low level until the required oscillation stabilization interval has

elapsed.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)		
орс	1	4	7F	_	_

Example: The statement

STOP NOP NOP NOP

halts all microcontroller operations.

SUB — Subtract

SUB dst,src

Operation: $dst \leftarrow dst - src$

The source operand is subtracted from the destination operand and the result is stored in the destination. The contents of the source are unaffected. Subtraction is performed by adding the two's complement of the source operand to the destination operand.

Flags: C: Set if a "borrow" occurred; cleared otherwise.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result is negative; cleared otherwise.

V: Set if arithmetic overflow occurred, that is, if the operands were of opposite signs and the sign of the result is of the same as the sign of the source operand; cleared otherwise.

D: Always set to "1".

H: Cleared if there is a carry from the most significant bit of the low-order four bits of the result; set otherwise indicating a "borrow".

Format:

			Bytes	Cycles	Opcode (Hex)	Addr <u>dst</u>	Mode <u>src</u>
орс	dst src		2	4	22	r	r
				6	23	r	lr
орс	src	dst	3	6	24	R	R
				6	25	R	IR
орс	dst	src	3	6	26	R	IM

Examples: Given: R1 = 12H, R2 = 03H, register 01H = 21H, register 02H = 03H, register 03H = 0AH:

In the first example, if working register R1 contains the value 12H and if register R2 contains the value 03H, the statement "SUB R1,R2" subtracts the source value (03H) from the destination value (12H) and stores the result (0FH) in destination register R1.

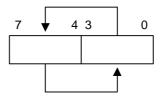


SWAP — Swap Nibbles

SWAP dst

Operation: $dst (0 - 3) \leftrightarrow dst (4 - 7)$

The contents of the lower four bits and upper four bits of the destination operand are swapped.



Flags: C: Undefined.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result bit 7 is set; cleared otherwise.

V: Undefined.

D: Unaffected.

H: Unaffected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u>
орс	dst	2	4	F0	R
			4	F1	IR

Examples: Given: Register 00H = 3EH, register 02H = 03H, and register 03H = 0A4H:

SWAP 00H \rightarrow Register 00H = 0E3H

SWAP @02H \rightarrow Register 02H = 03H, register 03H = 4AH

In the first example, if general register 00H contains the value 3EH (00111110B), the statement "SWAP 00H" swaps the lower and upper four bits (nibbles) in the 00H register, leaving the value 0E3H (11100011B).

TCM — Test Complement Under Mask

TCM dst,src

Operation: (NOT dst) AND src

This instruction tests selected bits in the destination operand for a logic one value. The bits to be tested are specified by setting a "1" bit in the corresponding position of the source operand (mask). The TCM statement complements the destination operand, which is then ANDed with the source mask. The zero (Z) flag can then be checked to determine the result. The destination and source operands are unaffected.

Flags: C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result bit 7 is set; cleared otherwise.

V: Always cleared to "0".

D: Unaffected.H: Unaffected.

Format:

			Ву	tes Cyc	cles Opco (Hex		Ir Mode <u>src</u>
орс	dst src		2	2 4	4 62	r	r
				(63	r	lr
орс	src	dst	(3 (6 64	R	R
				(65	R	IR
орс	dst	src	(3 (66	R	IM

Examples: Given: R0 = 0C7H, R1 = 02H, R2 = 12H, register 00H = 2BH, register 01H = 02H, and register 02H = 23H:

TCM R0,R1
$$\rightarrow$$
 R0 = 0C7H, R1 = 02H, Z = "1"

TCM R0,@R1 \rightarrow R0 = 0C7H, R1 = 02H, register 02H = 23H, Z = "0"

TCM 00H,01H \rightarrow Register 00H = 2BH, register 01H = 02H, Z = "1"

TCM 00H,@01H \rightarrow Register 00H = 2BH, register 01H = 02H, register 02H = 23H, Z = "1"

TCM 00H,#34 \rightarrow Register 00H = 2BH, Z = "0"

In the first example, if working register R0 contains the value 0C7H (11000111B) and register R1 the value 02H (00000010B), the statement "TCM R0,R1" tests bit one in the destination register for a "1" value. Because the mask value corresponds to the test bit, the Z flag is set to logic one and can be tested to determine the result of the TCM operation.



TM — Test Under Mask

TM dst,src

Operation: dst AND src

This instruction tests selected bits in the destination operand for a logic zero value. The bits to be tested are specified by setting a "1" bit in the corresponding position of the source operand (mask), which is ANDed with the destination operand. The zero (Z) flag can then be checked to determine the result. The destination and source operands are unaffected.

Flags: C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result bit 7 is set; cleared otherwise.

V: Always reset to "0".

D: Unaffected.

H: Unaffected.

Format:

			Byte	s Cycles	Opcode (Hex)	Addr <u>dst</u>	Mode <u>src</u>
орс	dst src		2	4	72	r	r
				6	73	r	lr
орс	src	dst	3	6	74	R	R
				6	75	R	IR
орс	dst	src	3	6	76	R	IM

Examples:

Given: R0 = 0C7H, R1 = 02H, R2 = 18H, register 00H = 2BH, register 01H = 02H, and register 02H = 23H:

TM R0,R1
$$\rightarrow$$
 R0 = 0C7H, R1 = 02H, Z = "0"

TM R0,@R1 \rightarrow R0 = 0C7H, R1 = 02H, register 02H = 23H, Z = "0"

TM 00H,01H \rightarrow Register 00H = 2BH, register 01H = 02H, Z = "0"

TM 00H,@01H \rightarrow Register 00H = 2BH, register 01H = 02H, register 02H = 23H, Z = "0"

TM 00H,#54H \rightarrow Register 00H = 2BH, Z = "1"

In the first example, if working register R0 contains the value 0C7H (11000111B) and register R1 the value 02H (00000010B), the statement "TM R0,R1" tests bit one in the destination register for a "0" value. Because the mask value does not match the test bit, the Z flag is cleared to logic zero and can be tested to determine the result of the TM operation.

WFI — Wait For Interrupt

WFI

Operation:

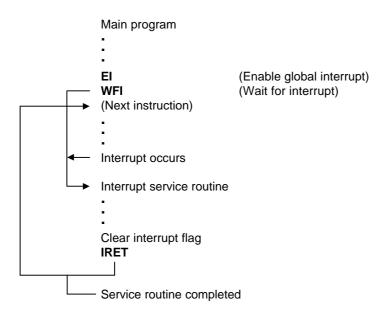
The CPU is effectively halted until an interrupt occurs, except that DMA transfers can still take place during this wait state. The WFI status can be released by an internal interrupt, including a fast interrupt .

Flags: No flags are affected.

Format:

·	Bytes	Cycles	Opcode (Hex)
орс	1	4n	3F
		(n = 1, 2, 3)	3,)

Example: The following sample program structure shows the sequence of operations that follow a "WFI" statement:



XOR — Logical Exclusive OR

XOR dst,src

Operation: $dst \leftarrow dst XOR src$

The source operand is logically exclusive-ORed with the destination operand and the result is stored in the destination. The exclusive-OR operation results in a "1" bit being stored whenever the corresponding bits in the operands are different; otherwise, a "0" bit is stored.

Flags: C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result bit 7 is set; cleared otherwise.

V: Always reset to "0".

D: Unaffected.H: Unaffected.

Format:

					Bytes	Cycles	Opcode (Hex)	Addr Mode <u>dst</u> <u>src</u>	
оро		dst src			2	4	B2	r	r
						6	B3	r	lr
				1					
оро	2	src	dst		3	6	B4	R	R
						6	B5	R	IR
оро	2	dst	src		3	6	B6	R	IM

Examples:

Given: R0 = 0C7H, R1 = 02H, R2 = 18H, register 00H = 2BH, register 01H = 02H, and register 02H = 23H:

```
XOR
         R0,R1
                            R0 = 0C5H, R1 = 02H
                            R0 = 0E4H, R1 = 02H, register 02H = 23H
XOR
         R0,@R1
XOR
                            Register 00H = 29H, register 01H = 02H
         00H,01H
                     \rightarrow
XOR
         00H,@01H
                            Register 00H = 08H, register 01H = 02H, register 02H = 23H
XOR
         00H,#54H
                            Register 00H = 7FH
```

In the first example, if working register R0 contains the value 0C7H and if register R1 contains the value 02H, the statement "XOR R0,R1" logically exclusive-ORs the R1 value with the R0 value and stores the result (0C5H) in the destination register R0.