6 SAM87RI INSTRUCTION SET

OVERVIEW

The SAM87RI instruction set is designed to support the large register file. It includes a full complement of 8-bit arithmetic and logic operations. There are 41 instructions. No special I/O instructions are necessary because I/O control and data registers are mapped directly into the register file. Flexible instructions for bit addressing, rotate, and shift operations complete the powerful data manipulation capabilities of the SAM87RI instruction set.

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 13-bit program memory or data memory addresses. For detailed information about register addressing, please refer to Section 2, "Address Spaces".

ADDRESSING MODES

There are six addressing modes: Register (R), Indirect Register (IR), Indexed (X), Direct (DA), Relative (RA), and Immediate (IM). 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
LDC	dst,src	Load program memory
LDE	dst,src	Load external data memory
LDCD	dst,src	Load program memory and decrement
LDED	dst,src	Load external data memory and decrement
LDCI	dst,src	Load program memory and increment
LDEI	dst,src	Load external data memory and increment
POP	dst	Pop from stack
PUSH	src	Push to stack
Arithmetic Instructions		
ADC	dst,src	Add with carry
ADD	dst,src	Add
CP	dst,src	Compare
DEC	dst	Decrement
INC	dst	Increment
SBC	dst,src	Subtract with carry
SUB	dst,src	Subtract
Logic Instructions		
AND	dst,src	Logical AND
COM	dst	Complement
OR	dst,src	Logical OR
XOR	dst,src	Logical exclusive OR

Table 6-1. Instruction Group Summary (Continued)

Mnemonic	Operands	Instruction
Program Control Instructions		
CALL	dst	Call procedure
IRET		Interrupt return
JP	cc,dst	Jump on condition code
JP	dst	Jump unconditional
JR	cc,dst	Jump relative on condition code
RET		Return
Bit Manipulation Instructions		
TCM	dst,src	Test complement under mask
TM	dst,src	Test under mask
Rotate and Shift Instructions		
RL	dst	Rotate left
RLC	dst	Rotate left through carry
RR	dst	Rotate right
RRC	dst	Rotate right through carry
SRA	dst	Shift right arithmetic
CPU Control Instructions		
CCF		Complement carry flag
DI		Disable interrupts
EI		Enable interrupts
IDLE		Enter Idle mode
NOP		No operation
RCF		Reset carry flag
SCF		Set carry flag
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.4–FLAGS.7, can be tested and used with conditional jump instructions.

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.

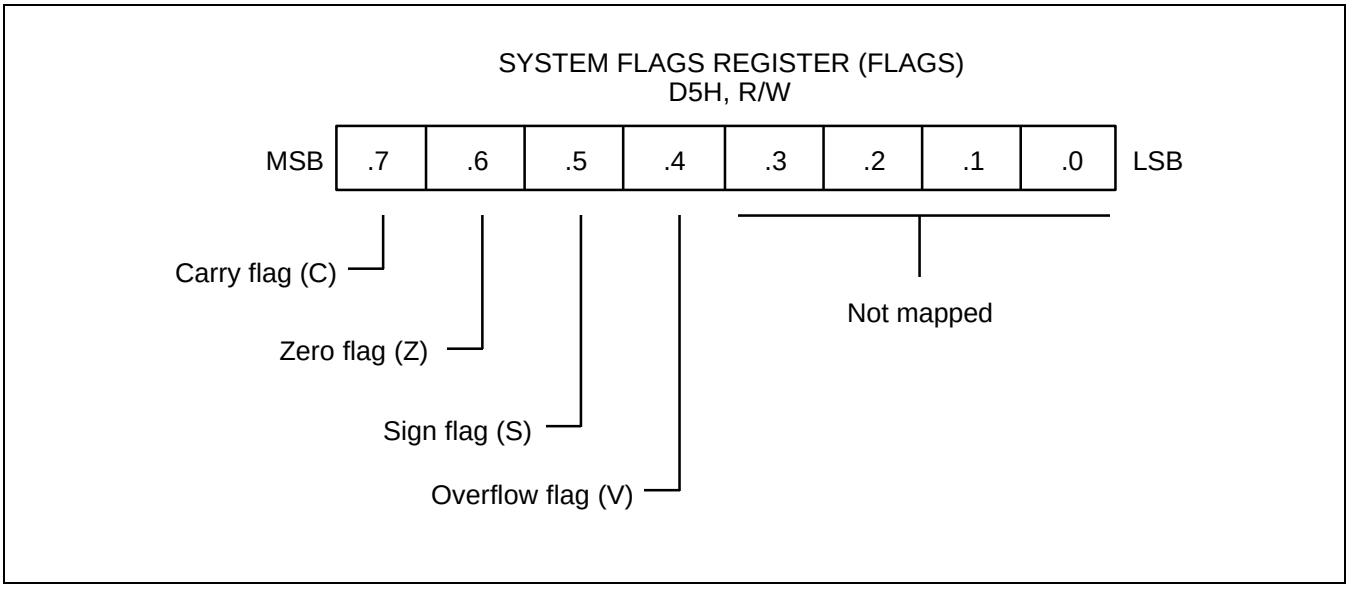


Figure 6-1. System Flags Register (FLAGS)

FLAG DESCRIPTIONS

Overflow Flag (FLAGS.4, V)

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.

Sign Flag (FLAGS.5, S)

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.

Zero Flag (FLAGS.6, Z)

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.

Carry Flag (FLAGS.7, C)

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.

INSTRUCTION SET NOTATION

Table 6-2. Flag Notation Conventions

Flag	Description
C	Carry flag
Z	Zero flag
S	Sign flag
V	Overflow 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
FLAGS	Flags register (D5H)
#	Immediate operand or register address prefix
H	Hexadecimal number suffix
D	Decimal number suffix
B	Binary number suffix
opc	Opcode

Table 6-4. Instruction Notation Conventions

Notation	Description	Actual Operand Range
cc	Condition code	See list of condition codes in Table 6-6.
r	Working register only	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)
RR	Register pair or working register pair	reg or RRp (reg = 0–254, even number only, where p = 0, 2, ..., 14)
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)
X	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 0–8191, where p = 0, 2, ..., 14)
DA	Direct addressing mode	addr (addr = range 0–8191)
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)

Table 6-5. Opcode Quick Reference

OPCODE MAP									
	LOWER NIBBLE (HEX)								
	–	0	1	2	3	4	5	6	7
U	0	DEC R1	DEC IR1	ADD r1,r2	ADD r1,Ir2	ADD R2,R1	ADD IR2,R1	ADD R1,IM	
P	1	RLC R1	RLC IR1	ADC r1,r2	ADC r1,Ir2	ADC R2,R1	ADC IR2,R1	ADC R1,IM	
P	2	INC R1	INC IR1	SUB r1,r2	SUB r1,Ir2	SUB R2,R1	SUB IR2,R1	SUB R1,IM	
E	3	JP IRR1		SBC r1,r2	SBC r1,Ir2	SBC R2,R1	SBC IR2,R1	SBC R1,IM	
R	4			OR r1,r2	OR r1,Ir2	OR R2,R1	OR IR2,R1	OR R1,IM	
	5	POP R1	POP IR1	AND r1,r2	AND r1,Ir2	AND R2,R1	AND IR2,R1	AND R1,IM	
N	6	COM R1	COM IR1	TCM r1,r2	TCM r1,Ir2	TCM R2,R1	TCM IR2,R1	TCM R1,IM	
I	7	PUSH R2	PUSH IR2	TM r1,r2	TM r1,Ir2	TM R2,R1	TM IR2,R1	TM R1,IM	
B	8								LD r1, x, r2
B	9	RL R1	RL IR1						LD r2, x, r1
L	A			CP r1,r2	CP r1,Ir2	CP R2,R1	CP IR2,R1	CP R1,IM	LDC r1, Irr2, xL
E	B	CLR R1	CLR IR1	XOR r1,r2	XOR r1,Ir2	XOR R2,R1	XOR IR2,R1	XOR R1,IM	LDC r2, Irr2, xL
	C	RRC R1	RRC IR1		LDC r1,Irr2				LD r1, Ir2
H	D	SRA R1	SRA IR1		LDC r2,Irr1			LD IR1,IM	LD Ir1, r2
E	E	RR R1	RR IR1	LDCD r1,Irr2	LDCI r1,Irr2	LD R2,R1	LD R2,IR1	LD R1,IM	LDC r1, Irr2, xs
X	F					CALL IRR1	LD IR2,R1	CALL DA1	LDC r2, Irr1, xs

Table 6-5. Opcode Quick Reference (Continued)

OPCODE MAP									
LOWER NIBBLE (HEX)									
	–	8	9	A	B	C	D	E	F
U	0	LD r1,R2	LD r2,R1		JR cc,RA	LD r1,IM	JP cc,DA	INC r1	
P	1	↓	↓		↓	↓	↓	↓	
P	2								
E	3								
R	4								
	5								
N	6								IDLE
I	7	↓	↓		↓	↓	↓	↓	STOP
B	8								DI
B	9								EI
L	A								RET
E	B								IRET
	C								RCF
H	D	↓	↓		↓	↓	↓	↓	SCF
E	E								CCF
X	F	LD r1,R2	LD r2,R1		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	T	Always true	–
0111 *	C	Carry	C = 1
1111 *	NC	No carry	C = 0
0110 *	Z	Zero	Z = 1
1110 *	NZ	Not zero	Z = 0
1101	PL	Plus	S = 0
0101	MI	Minus	S = 1
0100	OV	Overflow	V = 1
1100	NOV	No overflow	V = 0
0110 *	EQ	Equal	Z = 1
1110 *	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 *	UGE	Unsigned greater than or equal	C = 0
0111 *	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. Asterisks (*) indicate 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 SAM87RI 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 ← 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 Mode dst	Addr Mode src
opc | dst | src	2	6	12	r	r
			13	r	lr
opc | src | dst	3	10	14	R	R
			15	R	IR
opc | dst | src	3	10	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 → R1 = 14H, R2 = 03H
ADC R1,@R2 → R1 = 1BH, R2 = 03H
ADC 01H,02H → Register 01H = 24H, register 02H = 03H
ADC 01H,@02H → Register 01H = 2BH, register 02H = 03H
ADC 01H,#11H → 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 ← 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 Mode dst	Addr Mode src
opc | dst | src	2	6	02	r	r
			03	r	lr
opc | src | dst	3	10	04	R	R
			05	R	IR
opc | dst | src	3	10	06	R	IM

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

ADD R1,R2 → R1 = 15H, R2 = 03H
ADD R1,@R2 → R1 = 1CH, R2 = 03H
ADD 01H,02H → Register 01H = 24H, register 02H = 03H
ADD 01H,@02H → Register 01H = 2BH, register 02H = 03H
ADD 01H,#25H → Register 01H = 46H

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 ← 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 Mode dst	Addr Mode src
opc | dst | src	2	6	52	r	r
			53	r	lr
opc | src | dst	3	10	54	R	R
			55	R	IR
opc | dst | src	3	10	56	R	IM

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

AND R1,R2 → R1 = 02H, R2 = 03H
AND R1,@R2 → R1 = 02H, R2 = 03H
AND 01H,02H → Register 01H = 01H, register 02H = 03H
AND 01H,@02H → Register 01H = 00H, register 02H = 03H
AND 01H,#25H → 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.

CALL — Call Procedure

CALL dst

Operation:
SP ← SP – 1
@SP ← PCL
SP ← SP –1
@SP ← PCH
PC ← dst

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 dst
opc | dst	3	18	F6	DA
opc | dst	2	18	F4	IRR

Examples: Given: R0 = 15H, R1 = 21H, PC = 1A47H, and SP = 0B2H:

CALL 1521H → SP = 0B0H
(Memory locations 00H = 1AH, 01H = 4AH, where 4AH is the address that follows the instruction.)

CALL @RR0 → SP = 0B0H (00H = 1AH, 01H = 49H)

In the first example, if the program counter value is 1A47H and the stack pointer contains the value 0B2H, the statement "CALL 1521H" pushes the current PC value onto the top of the stack. The stack pointer now points to memory location 00H. The PC is then loaded with the value 1521H, 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 01H (because the two-byte instruction format was used). The PC is then loaded with the value 1521H, the address of the first instruction in the program sequence to be executed.

CCF — Complement Carry Flag

CCF

Operation: C ← 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)
opc	1	6	EF

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 ← "0"

The destination location is cleared to "0".

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc | dst	2	6	B0	R
			B1	IR

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

CLR 00H → Register 00H = 00H

CLR @01H → 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.

COM — Complement

COM dst

Operation: dst ← 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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc | dst	2	6	60	R
			61	IR

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

COM R1 → R1 = 0F8H
COM @R1 → 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.

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 signs and the sign of the result is of the same as the sign of the source operand; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc | dst | src	2	6	A2	r	r
			A3	r	lr
opc | src | dst	3	10	A4	R	R
			A5	R	IR
opc | dst | src	3	10	A6	R	IM

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

```
CP      R1,R2  →    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:

```
        CP      R1,R2
        JP      UGE,SKIP
        INC     R1
SKIP    LD      R3,R1
```

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.

DEC — Decrement

DEC dst

Operation: dst ← 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, that is, dst value is –128 (80H) and result value is +127(7FH); cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc | dst	2	6	00	R
			01	IR

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

DEC R1 → R1 = 02H
DEC @R1 → 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.

DI — Disable Interrupts

DI

Operation: SYM (2) ← 0

Bit zero of the system mode register, SYM.2, 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)
opc	1	6	8F

Example: Given: SYM = 04H:

DI

If the value of the SYM register is 04H, the statement "DI" leaves the new value 00H in the register and clears SYM.2 to "0", disabling interrupt processing.

EI — Enable Interrupts

EI

Operation: SYM (2) ← 1

An EI instruction sets bit 2 of the system mode register, SYM.2 to "1". This allows interrupts to be serviced as they occur. 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)
opc	1	6	9F

Example: Given: SYM = 00H:

EI

If the SYM register contains the value 00H, that is, if interrupts are currently disabled, the statement "EI" sets the SYM register to 04H, enabling all interrupts (SYM.2 is the enable bit for global interrupt processing).

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)	Addr Mode dst	src
opc	1	3	6F	–	–

Example: The instruction

IDLE

stops the CPU clock but not the system clock.

INC — Increment

INC dst

Operation: dst ← 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, that is dst value is +127 (7FH) and result is –128 (80H); cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
dst | opc	1	6	rE r = 0 to F	r
opc | dst	2	6	20	R
			21	IR

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

INC R0 → R0 = 1CH

INC 00H → Register 00H = 0DH

INC @R0 → 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.

IRET — Interrupt Return

IRET IRET

Operation: FLAGS ← @SP
SP ← SP + 1
PC ← @SP
SP ← SP + 2
SYM(2) ← 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.

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

Format:

IRET (Normal)	Bytes	Cycles	Opcode (Hex)
opc	1	16	BF

JP — Jump

JP cc,dst (Conditional)

JP dst (Unconditional)

Operation: If cc is true, PC ← 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 dst
cc | opc | dst	3	10/12 (3)	ccD cc = 0 to F	DA
opc | dst	2	10	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.
3. For a conditional jump, execution time is 12 cycles if the jump is taken or 10 cycles if it is not taken.

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

JP C,LABEL_W → LABEL_W = 1000H, PC = 1000H

JP @00H → 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 ← 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)

cc | opc	dst

Bytes	Cycles	Opcode (Hex)	Addr Mode dst
2	10/12 (2)	ccB	RA
		cc = 0 to F	

NOTES:
1. In the first byte of the two-byte instruction format, the condition code and the opcode are each four bits.
2. Instruction execution time is 12 cycles if the jump is taken or 10 cycles if it is not taken.

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

JR C,LABEL_X → 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 ← 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 Mode dst	src
dst | opc / src	2	6	rC	r	IM
		6	r8	r	R
src | opc / dst	2	6	r9	R	r
			r = 0 to F		
opc / dst | src	2	6	C7	r	lr
		6	D7	lr	r
opc / src / dst	3	10	E4	R	R
		10	E5	R	IR
opc / dst / src	3	10	E6	R	IM
		10	D6	IR	IM
opc / src / dst	3	10	F5	IR	R
opc / dst | src / x	3	10	87	r	x [r]
opc / src | dst / x	3	10	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 → R0 = 10H

LD R0,01H → R0 = 20H, register 01H = 20H

LD 01H,R0 → Register 01H = 01H, R0 = 01H

LD R1,@R0 → R1 = 20H, R0 = 01H

LD @R0,R1 → R0 = 01H, R1 = 0AH, register 01H = 0AH

LD 00H,01H → Register 00H = 20H, register 01H = 20H

LD 02H,@00H → Register 02H = 20H, register 00H = 01H

LD 00H,#0AH → Register 00H = 0AH

LD @00H,#10H → Register 00H = 01H,

register 01H = 10H

LD @00H,02H → Register 00H = 01H, register 01H = 02, register 02H = 02H

LD R0,#LOOP[R1] → R0 = 0FFH, R1 = 0AH

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

LDC/LDE — Load Memory

LDC/LDE dst,src

Operation: dst ← 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 Mode dst	Addr Mode src
1.	opc	dst | src			2	12	C3	r	Irr
2.	opc	src | dst			2	12	D3	Irr	r
3.	opc	dst | src	XS		3	18	E7	r	XS [rr]
4.	opc	src | dst	XS		3	18	F7	XS [rr]	r
5.	opc	dst | src	XL_L	XL_H	4	20	A7	r	XL [rr]
6.	opc	src | dst	XL_L	XL_H	4	20	B7	XL [rr]	r
7.	opc	dst | 0000	DA_L	DA_H	4	20	A7	r	DA
8.	opc	src | 0000	DA_L	DA_H	4	20	B7	DA	r
9.	opc	dst | 0001	DA_L	DA_H	4	20	A7	r	DA
10.	opc	src | 0001	DA_L	DA_H	4	20	B7	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, R4 = 00H, R5 = 60H; Program memory locations 0061 = AAH, 0103H = 4FH, 0104H = 1A, 0105H = 6DH, and 1104H = 88H. External data memory locations 0061H = BBH, 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 *   @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 → no change

LDC R0,#01H[RR4] ; R0 ← contents of program memory location 0061H

; (01H + RR4),

; R0 = AAH, R2 = 00H, R3 = 60H

LDE R0,#01H[RR4] ; R0 ← contents of external data memory location 0061H

; (01H + RR4), R0 = BBH, R4 = 00H, R5 = 60H

LDC * #01H[RR4],R0 ; 11H (contents of R0) is loaded into program memory location

; 0061H (01H + 0060H)

LDE #01H[RR4],R0 ; 11H (contents of R0) is loaded into external data memory

; location 0061H (01H + 0060H)

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

; 1105H, (1105H) ← 11H

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

; location 1105H, (1105H) ← 11H

* These instructions are not supported by masked ROM type devices.

LDCD/LDED — Load Memory and Decrement

LDCD/LDED dst,src

Operation: dst ← src

rr ← 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 (Hex)	Addr Mode dst	Addr Mode src
opc | dst | src	2	16	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 ← RR6 – 1)
LDED R8,@RR6    ;   0DDH (contents of data memory location 1033H) is loaded
```

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

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

LDCI/LDEI — Load Memory and Increment

LDCI/LDEI dst,src

Operation: dst ← src

rr ← 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 (Hex)	Addr Mode dst	src
opc	dst | src	2	16	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 ← 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 ← RR6 + 1)

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

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)
opc	1	6	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 ← 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.

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 Mode dst	src
opc | dst | src	2	6	42	r	r
		6	43	r	lr
opc | src | dst	3	10	44	R	R
		10	45	R	IR
opc | dst | src	3	10	46	R	IM

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

OR R0,R1 → R0 = 3FH, R1 = 2AH

OR R0,@R2 → R0 = 37H, R2 = 01H, register 01H = 37H

OR 00H,01H → Register 00H = 3FH, register 01H = 37H

OR 01H,@00H → Register 00H = 08H, register 01H = 0BFH

OR 00H,#02H → 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 ← @SP

SP ← 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:

opc	dst

Bytes	Cycles	Opcode (Hex)	Addr Mode dst
2	10	50	R
	10	51	IR

Examples: Given: Register 00H = 01H, register 01H = 1BH, SP (0D9H) = 0BBH, and stack register 0BBH = 55H:

POP 00H → Register 00H = 55H, SP = 0BCH

POP @00H → Register 00H = 01H, register 01H = 55H, SP = 0BCH

In the first example, general register 00H contains the value 01H. The statement "POP 00H" loads the contents of location 0BBH (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 0BCH.

PUSH — Push to Stack

PUSH src

Operation: SP ← SP – 1

@SP ← 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 dst
opc src	2	10	70	R
		12	71	IR

Examples: Given: Register 40H = 4FH, register 4FH = 0AAH, SP = 0C0H:

PUSH 40H → Register 40H = 4FH, stack register 0BFH = 4FH,

SP = 0BFH

PUSH @40H → Register 40H = 4FH, register 4FH = 0AAH, stack register 0BFH = 0AAH, SP = 0BFH

In the first example, if the stack pointer contains the value 0C0H, and general register 40H the value 4FH, the statement "PUSH 40H" decrements the stack pointer from 0C0 to 0BFH. It then loads the contents of register 40H into location 0BFH. Register 0BFH then contains the value 4FH and SP points to location 0BFH.

RCF — Reset Carry Flag

RCF RCF

Operation: C ← 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	6	CF

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

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

RET — Return

RET

Operation: PC ← @SP

SP ← 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)
opc	1	14	AF

Example: Given: SP = 0BCH, (SP) = 101AH, and PC = 1234:

RET → PC = 101AH, SP = 0BEH

The statement "RET" pops the contents of stack pointer location 0BCH (10H) into the high byte of the program counter. The stack pointer then pops the value in location 0BDH (1AH) into the PC's low byte and the instruction at location 101AH is executed. The stack pointer now points to memory location 0BEH.

RL — Rotate Left

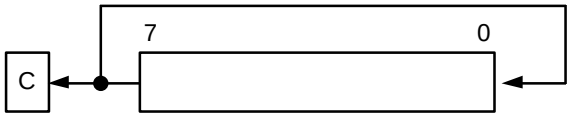
RL dst

Operation: C ← dst (7)

dst (0) ← dst (7)

dst (n + 1) ← 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, that is, if the sign of the destination changed during rotation; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc | dst	2	6	90	R
		6	91	IR

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

RL 00H → Register 00H = 55H, C = "1"

RL @01H → 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

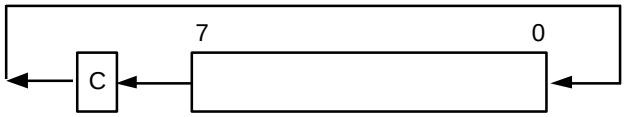
RLC dst

Operation: dst (0) ← C

C ← dst (7)

dst (n + 1) ← 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 dst
opc dst	2	6	10	R
		6	11	IR

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

RLC 00H → Register 00H = 54H, C = "1"

RLC @01H → 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

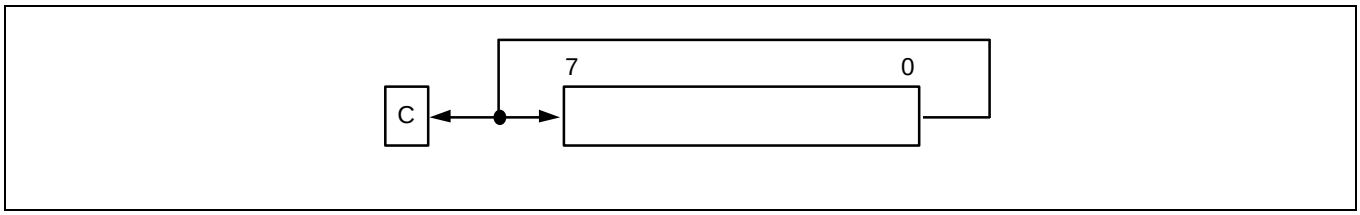
RR dst

Operation: C ← dst (0)

dst (7) ← dst (0)

dst (n) ← 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 dst
opc dst	2	6	E0	R
		6	E1	IR

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

RR 00H → Register 00H = 98H, C = "1"

RR @01H → 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

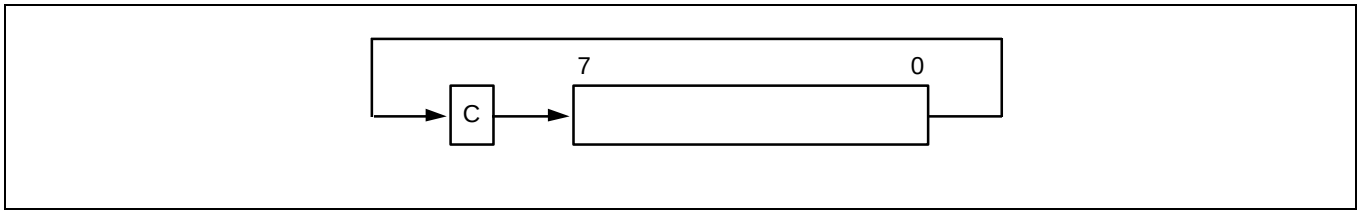
RRC dst

Operation: dst (7) ← C

C ← dst (0)

dst (n) ← 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 dst
opc | dst	2	6	C0	R
		6	C1	IR

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

RRC 00H → Register 00H = 2AH, C = "1"

RRC @01H → 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".

SBC — Subtract with Carry

SBC dst,src

Operation: dst ← 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	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	dst | src		2	6	32	r	r
				6	33	r	lr
opc	src	dst	3	10	34	R	R
				10	35	R	IR
opc	dst	src	3	10	36	R	IM

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

SBC R1,R2 → R1 = 0CH, R2 = 03H

SBC R1,@R2 → R1 = 05H, R2 = 03H, register 03H = 0AH

SBC 01H,02H → Register 01H = 1CH,

register 02H = 03H

SBC 01H,@02H → Register 01H = 15H, register 02H = 03H, register 03H = 0AH

SBC 01H,#8AH → 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 ← 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)
opc	1	6	DF

Example: The statement

SCF

sets the carry flag to logic one.

SRA — Shift Right Arithmetic

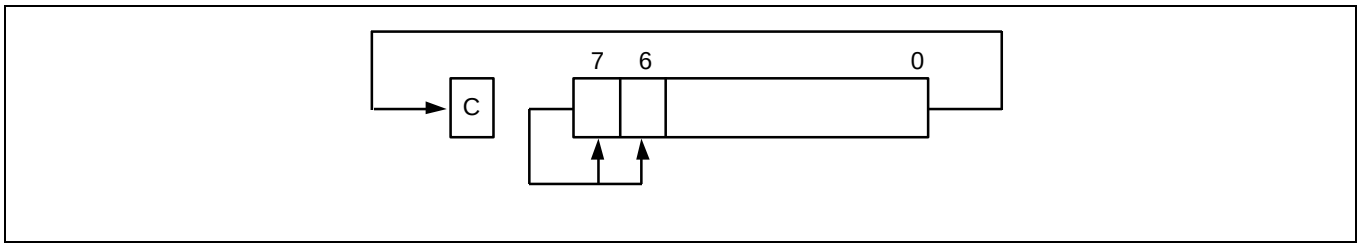
SRA dst

Operation: dst (7) ← dst (7)

C ← dst (0)

dst (n) ← 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 dst
opc | dst	2	6	D0	R
		6	D1	IR

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

SRA 00H → Register 00H = 0CD, C = "0"

SRA @02H → 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.

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 External interrupt input. 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)	Addr Mode dst	src
opc	1	3	7F	–	–

Example: The statement

STOP

halts all microcontroller operations.

SUB — Subtract

SUB dst,src

Operation: dst ← 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 Mode dst	Addr Mode src
opc | dst | src	2	6	22	r	r
		6	23	r	lr
opc | src | dst	3	10	24	R	R
		10	25	R	IR
opc | dst | src	3	10	26	R	IM

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

SUB R1,R2 → R1 = 0FH, R2 = 03H

SUB R1,@R2 → R1 = 08H, R2 = 03H

SUB 01H,02H → Register 01H = 1EH, register 02H = 03H

SUB 01H,@02H → Register 01H = 17H,

register 02H = 03H

SUB 01H,#90H → Register 01H = 91H; C, S, and V = "1"

SUB 01H,#65H → Register 01H = 0BCH; C and S = "1", V = "0"

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.

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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc | dst | src	2	6	62	r	r
		6	63	r	lr
opc | src | dst	3	10	64	R	R
		10	65	R	IR
opc | dst | src	3	10	66	R	IM

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

TCM R0,R1 → R0 = 0C7H, R1 = 02H, Z = "1"

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

TCM 00H,01H → Register 00H = 2BH,

register 01H = 02H, Z = "1"

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

TCM 00H,#34 → 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:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc | dst | src	2	6	72	r	r
		6	73	r	lr
opc | src | dst	3	10	74	R	R
		10	75	R	IR
opc | dst | src	3	10	76	R	IM

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

TMR0,R1 → R0 = 0C7H, R1 = 02H, Z = "0"

TMR0,@R1 → R0 = 0C7H, R1 = 02H, register 02H = 23H, Z = "0"

TM00H,01H → Register 00H = 2BH, register 01H = 02H, Z = "0"

TM00H,@01H → Register 00H = 2BH, register 01H = 02H, register 02H = 23H, Z = "0"

TM 00H,#54H → 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.

XOR — Logical Exclusive OR

XOR dst,src

Operation: dst ← 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 dst	src
opc | dst | src	2	6	B2	r	r
		6	B3	r	lr
opc | src | dst	3	10	B4	R	R
		10	B5	R	IR
opc | dst | src	3	10	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

XOR R0,@R1 → R0 = 0E4H, R1 = 02H, register 02H = 23H

XOR 00H,01H → Register 00H = 29H, register 01H = 02H

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.