SONiX 8-Bit MCU

INSTRUCTION SET

General Release Specification

SONiX 8-Bit Micro-Controller

AMENDENT HISTORY

Version	**Date**	**Description**
VER 1.90	Sep. 2002	V1.90 first issue
VER 1.93	Feb. 2003	Change the description of ADD M,A instruction from “M $\leftarrow$ M+A” to “M $\leftarrow$ A+M”
VER 1.94	Jul. 2004	1. Separate instruction table into two tables. 2. Add PUSH/POP instruction description. 3. Remove the instruction cycle description in each instruction description.

Table of Content

1 OVERVIEW

This Manual contains detailed information and programming examples for each instruction of SONiX 8-Bit MCU series instruction set. Information is arranged in a consistent format to improve readability and for use as a quick-reference resource for application programmers.

The information elements of the instruction description format are as follows:

— Instruction name (mnemonic)
— Full instruction name
— Operand overview
— Source/destination format of the instruction operand
— Textual description of instructions
— Programming examples to shoe how the instruction is used

2 INSTRUCTION SET TABLE

Table I: For SN8P1XXX (Four clocks per instruction cycle)

Field	Mnemonic		Description	C	DC	Z	Cycle
	MOV	A,M	A ← M	-	-	√	1
M	MOV	M,A	M ← A	-	-	-	1
O	B0MOV	A,M	A ← M (bnak 0)	-	-	√	1
V	B0MOV	M,A	M (bank 0) ← A	-	-	-	1
E	MOV	A,I	A ← I	-	-	-	1
	B0MOV	M,I	M ← I, (M = only for Working registers R, Y, Z , RBANK & PFLAG)	-	-	-	1
	XCH	A,M	A ←→M	-	-	-	1
	B0XCH	A,M	A ←→M (bank 0)	-	-	-	1
	MOVC		R, A ← ROM [Y,Z]	-	-	-	2
	ADC	A,M	A ← A + M + C, if occur carry, then C=1, else C=0	√	√	√	1
A	ADC	M,A	M ← A + M + C, if occur carry, then C=1, else C=0	√	√	√	1
R	ADD	A,M	A ← A + M, if occur carry, then C=1, else C=0	√	√	√	1
I	ADD	M,A	M ← A + M, if occur carry, then C=1, else C=0	√	√	√	1
T	B0ADD	M,A	M (bank 0) ← A + M (bank 0) , if occur carry, then C=1, else C=0	√	√	√	1
H	ADD	A,I	A ← A + I, if occur carry, then C=1, else C=0	√	√	√	1
M	SBC	A,M	A ← A - M - /C, if occur borrow, then C=0, else C=1	√	√	√	1
E	SBC	M,A	M ← A - M - /C, if occur borrow, then C=0, else C=1	√	√	√	1
T	SUB	A,M	A ← A - M, if occur borrow, then C=0, else C=1	√	√	√	1
I	SUB	M,A	M ← A - M, if occur borrow, then C=0, else C=1	√	√	√	1
C	SUB	A,I	A ← A - I, if occur borrow, then C=0, else C=1	√	√	√	1
	DAA		To adjust ACC’s data format from HEX to DEC.	√	-	-	1
	MUL	A,M	R, A ← A * M, The LB of product stored in Acc and HB stored in R register. ZF affected by Acc.	-	-	√	2
	AND	A,M	A ← A and M	-	-	√	1
L	AND	M,A	M ← A and M	-	-	√	1
O	AND	A,I	A ← A and I	-	-	√	1
G	OR	A,M	A ← A or M	-	-	√	1
I	OR	M,A	M ← A or M	-	-	√	1
C	OR	A,I	A ← A or I	-	-	√	1
	XOR	A,M	A ← A xor M	-	-	√	1
	XOR	M,A	M ← A xor M	-	-	√	1
	XOR	A,I	A ← A xor I	-	-	√	1
	SWAP	M	A (b3~b0, b7~b4) ←M(b7~b4, b3~b0)	-	-	-	1
P	SWAPM	M	M(b3~b0, b7~b4) ← M(b7~b4, b3~b0)	-	-	-	1
R	RRC	M	A ← RRC M	√	-	-	1
O	RRCM	M	M ← RRC M	√	-	-	1
C	RLC	M	A ← RLC M	√	-	-	1
E	RLCM	M	M ← RLC M	√	-	-	1
S	CLR	M	M ← 0	-	-	-	1
S	BCLR	M.b	M.b ← 0	-	-	-	1
	BSET	M.b	M.b ← 1	-	-	-	1
	B0BCLR	M.b	M(bank 0).b ← 0	-	-	-	1
	B0BSET	M.b	M(bank 0).b ← 1	-	-	-	1

Field	Mnemonic	Description	C	DC	Z	Cycle
BRANCH	CMPRS A,I	ZF,C ← A - I, If A = I, then skip next instruction	√	-	√	1 + S
	CMPRS A,M	ZF,C ← A – M, If A = M, then skip next instruction	√	-	√	1 + S
	INCS M	A ← M + 1, If A = 0, then skip next instruction	-	-	-	1 + S
	INCMS M	M ← M + 1, If M = 0, then skip next instruction	-	-	-	1 + S
	DECS M	A ← M - 1, If A = 0, then skip next instruction	-	-	-	1 + S
	DECMS M	M ← M - 1, If M = 0, then skip next instruction	-	-	-	1 + S
	BTS0 M.b	If M.b = 0, then skip next instruction	-	-	-	1 + S
	BTS1 M.b	If M.b = 1, then skip next instruction	-	-	-	1 + S
	B0BTS0 M.b	If M(bank 0).b = 0, then skip next instruction	-	-	-	1 + S
	B0BTS1 M.b	If M(bank 0).b = 1, then skip next instruction	-	-	-	1 + S
	JMP d	PC15/14 ← RomPages1/0, PC13~PC0 ← d	-	-	-	2
	CALL d	Stack ← PC15~PC0, PC15/14 ← RomPages1/0, PC13~PC0 ← d	-	-	-	2
MISC	RET	PC ← Stack	-	-	-	2
	RETI	PC ← Stack, and to enable global interrupt	-	-	-	2
	RETLW	PC ← Stack, and to load a value by PC+A	-	-	-	2
	PUSH	To push working registers (080H~087H) into buffers	-	-	-	1
	POP	To pop working registers (080H~087H) from buffers	√	√	√	1
	NOP	No operation	-	-	-	1

Remark

1. The “M” is memory including system registers and user defined memory.
2. If branch condition is true then “S = 0”, otherwise “S = 1”.
3. Any instruction that writes OSCM register will add an extra cycle.
4. One instruction cycle = 1/Fcpu

Table II: For SN8P2XXX (One clock per instruction cycle)

Field	Mnemonic		Description	C	DC	Z	Cycle
MOVE	MOV	A,M	A ← M	-	-	√	1
	MOV	M,A	M ← A	-	-	-	1
	B0MOV	A,M	A ← M (bnak 0)	-	-	√	1
	B0MOV	M,A	M (bank 0) ← A	-	-	-	1
	MOV	A,I	A ← I	-	-	-	1
	B0MOV	M,I	M ← I, (M = Working registers, RBANK & PFLAG)	-	-	-	1
	XCH	A,M	A ←→M	-	-	-	**1+N**
	B0XCH	A,M	A ←→M (bank 0)	-	-	-	**1+N**
	MOVC		R, A ← ROM [Y,Z]	-	-	-	2
ARITHMETIC	ADC	A,M	A ← A + M + C, if occur carry, then C=1, else C=0	√	√	√	1
	ADC	M,A	M ← A + M + C, if occur carry, then C=1, else C=0	√	√	√	**1+N**
	ADD	A,M	A ← A + M, if occur carry, then C=1, else C=0	√	√	√	1
	ADD	M,A	M ← M + A, if occur carry, then C=1, else C=0	√	√	√	**1+N**
	B0ADD	M,A	M (bank 0) ← M (bank 0) + A, if occur carry, then C=1, else C=0	√	√	√	**1+N**
	ADD	A,I	A ← A + I, if occur carry, then C=1, else C=0	√	√	√	1
	SBC	A,M	A ← A - M - /C, if occur borrow, then C=0, else C=1	√	√	√	1
	SBC	M,A	M ← A - M - /C, if occur borrow, then C=0, else C=1	√	√	√	**1+N**
	SUB	A,M	A ← A - M, if occur borrow, then C=0, else C=1	√	√	√	1
	SUB	M,A	M ← A - M, if occur borrow, then C=0, else C=1	√	√	√	**1+N**
	SUB	A,I	A ← A - I, if occur borrow, then C=0, else C=1	√	√	√	1
	DAA		To adjust ACC's data format from HEX to DEC.	√	-	-	1
	MUL	A,M	R, A ← A * M, The LB of product stored in Acc and HB stored in R register. ZF affected by Acc.	-	-	√	2
LOGIC	AND	A,M	A ← A and M	-	-	√	1
	AND	M,A	M ← A and M	-	-	√	**1+N**
	AND	A,I	A ← A and I	-	-	√	1
	OR	A,M	A ← A or M	-	-	√	1
	OR	M,A	M ← A or M	-	-	√	**1+N**
	OR	A,I	A ← A or I	-	-	√	1
	XOR	A,M	A ← A xor M	-	-	√	1
	XOR	M,A	M ← A xor M	-	-	√	**1+N**
	XOR	A,I	A ← A xor I	-	-	√	1
PROCESS	SWAP	M	A (b3~b0, b7~b4) ←M(b7~b4, b3~b0)	-	-	-	1
	SWAPM	M	M(b3~b0, b7~b4) ← M(b7~b4, b3~b0)	-	-	-	**1+N**
	RRC	M	A ← RRC M	√	-	-	1
	RRCM	M	M ← RRC M	√	-	-	**1+N**
	RLC	M	A ← RLC M	√	-	-	1
	RLCM	M	M ← RLC M	√	-	-	**1+N**
	CLR	M	M ← 0	-	-	-	1
	BCLR	M.b	M.b ← 0	-	-	-	**1+N**
	BSET	M.b	M.b ← 1	-	-	-	**1+N**
	B0BCLR	M.b	M(bank 0).b ← 0	-	-	-	**1+N**
	B0BSET	M.b	M(bank 0).b ← 1	-	-	-	**1+N**

Field	Mnemonic	Description	C	DC	Z	Cycle
BRANCH	CMPRS A,I	ZF,C ← A - I, If A = I, then skip next instruction	√	-	√	1 + S
	CMPRS A,M	ZF,C ← A - M, If A = M, then skip next instruction	√	-	√	1 + S
	INCS M	A ← M + 1, If A = 0, then skip next instruction	-	-	-	1 + S
	INCMS M	M ← M + 1, If M = 0, then skip next instruction	-	-	-	**1+N+S**
	DECS M	A ← M - 1, If A = 0, then skip next instruction	-	-	-	1 + S
	DECMS M	M ← M - 1, If M = 0, then skip next instruction	-	-	-	**1+N+S**
	B0BTS0 M.b	If M(bank 0).b = 0, then skip next instruction	-	-	-	1 + S
	B0BTS1 M.b	If M(bank 0).b = 1, then skip next instruction	-	-	-	1 + S
	JMP D	PC15/14 ← RomPages1/0, PC13~PC0 ← d	-	-	-	2
	CALL D	Stack ← PC15~PC0, PC15/14 ← RomPages1/0, PC13~PC0 ← d	-	-	-	2
MIS	RET	PC ← Stack	-	-	-	2
	RETI	PC ← Stack, and to enable global interrupt	-	-	-	2
	PUSH	a. To push working registers (080H~087H) into buffers b. To push ACC and PFLAG (except NT0, NPD bit) into buffers. Note: Refer to instruction description section	-	-	-	1
	POP	a. To pop working registers (080H~087H) from buffers b. To pop ACC and PFLAG (except NT0, NPD bit) into buffers Note: Refer to instruction description section	√	√	√	1
	NOP	No operation	-	-	-	1
	RETLW I	PC← Stack, A← I	-	-	-	2

Remark

1. The “M” is memory including system registers and user defined memory.
2. If branch condition is true then “S = 0”, otherwise “S = 1”.
3. If “M” is system registers (80h ~ FFh of bank 0) then “N” = 0, otherwise “N” = 1
4. Different to SN8P1XXX series, any instruction that writes OSCM register **does not** add an extra cycle.
5. One instruction cycle = 1/Fcpu
6. Instruction limitations:
   - The ROM address 8 only JMP or NOP instruction is valid in following chips:
     SN8P2501A, SN8P2602A, SN8P2604, SN8P2606, SN8P2608, SN8P270XA
   - For "B0MOV M,I" instruction, the value of I can't be 0E6h or 0E7h in following chips:
     SN8P2501A, SN8P2602A, SN8P2604, SN8P270XA
     E.g. "B0MOV Y, #0E6h" is invalid
     "B0MOV Y, #00h" is valid
   - For "B0XCH A,M" instruction, the address range of M can't be 80h ~ FFh in following chips:
     SN8P2501A, SN8P2602A, SN8P2604, SN8P270XA
   - Read, modify then write back memory instruction can't access T0C register in following chips:
     SN8P2602A, SN8P2604, SN8P2606, SN8P2608, SN8P270XA
     E.g. "B0ADD T0C, A" is invalid
     "B0MOV T0C, A" is valid

INSTRUCTION DESCRIPTION

DATA TRANSFER INSTRUCTION

MOV – Memory Read / Write Instruction

➲ **Operation:**

Mnemonic		*Description*	*C*	*DC*	*Z*
MOV	A,M	A ← M	-	-	√
MOV	M,A	M ← A	-	-	-
MOV	A,I	A ← I	-	-	-

Remark

1. The "I" is immediately value.
2. The "M" is memory including system registers and user defined memory.

➲ **Description:**

The instruction is a memory read/write instruction. It transfers data through ACC. There are three operations of MOV instruction.

- ***MOV A,M : Read memory data and store into ACC buffer. If the result is zero, the zero flag (Z) will be set as "1". (Support R/W, R register and memory)***
- ***MOV M,A : Write the data from ACC buffer to memory. The operation doesn't affect PFLAG. (Support R/W, W register and memory)***
- ***MOV A,I : Write a immediate value to ACC buffer. The operation doesn't affect PFLAG.***

➲ **Example:**

```
MOV        A, #55H        ; Write 55H immediate value to ACC

MOV        WK00, A        ; Write ACC buffer data to WK00 defined by user.

MOV        A, WK00        ; Read WK00 data and store in to ACC buffer.
```

B0MOV – BANK 0 Memory Read / Write Instruction

➲ **Operation:**

Mnemonic	*Description*	*C*	*DC*	*Z*
B0MOV A,M	A ← M (bnak 0)	-	-	√
B0MOV M,A	M (bank 0) ← A	-	-	-
B0MOV M,I	M ← I, (M = Working registers, RBANK & PFLAG)	-	-	-

Remark

1. ***The "I" is immediately value.***
2. ***The "M" is BANK 0 memory including system registers and user defined memory.***

➲ **Description:**

The instruction is a memory read/write instruction. **The memory must be in BANK 0**. It transfers data through ACC. There are three operations of B0MOV instruction.

➢ ***B0MOV A,M : Read BANK 0 memory data and store into ACC buffer. If the result is zero, the zero flag (Z) will be set as "1"; otherwise, the zero flag (Z) is cleared. (Support R/W, R register and memory)***

➢ ***B0MOV M,A : Write the data from ACC buffer to BANK 0 memory. The operation doesn't affect PFLAG. (Support R/W, W register and memory)***

➢ ***B0MOV M,I : Write a immediate value to BANK 0 memory. The memory must be working register (H, L, R, X, Y, Z), RBANK or PFLAG. The operation doesn't affect PFLAG.***

➲ **Example:**

```
B0MOV     A, WK00      ; Read WK00 data and store into ACC. The WK00 is defined
                       by user
                       ; and in BANK 0.

B0MOV     WK00, A      ; Write ACC buffer data to WK00 defined by user.

B0MOV     R, #10       ; Write an immediate data into R register directly.
```

XCH – Data Exchange for All Area Memory

➲ **Operation:**

Mnemonic	Description	C	DC	Z
XCH A,M	A ←→M	-	-	-

Remark

The "M" is memory including system registers and user defined memory.

➲ **Description:**

The instruction is to exchange the data between ACC and memory. When the XCH is executed, the ACC data is transferred to the memory, and data of memory transferred to ACC buffer.

➢ ***XCH M,A : Exchange the data between ACC and the memory. The operation doesn't affect PFLAG. (Support R/W register and memory)***

➲ **Example:**

```
          XCH          A, WK00          ; Exchange the data between ACC and WK00 defined
                                        ; by user.
```

B0XCH – Data Exchange for BANK 0 Memory

➲ **Operation:**

Mnemonic	Description	C	DC	Z
B0XCH A,M	A ←→M (bank 0)	-	-	-

Remark

The "M" is BANK 0 memory including system registers and user defined memory.

➲ **Description:**

The instruction is to exchange the data between ACC and memory. **The memory must be in BANK 0.** When the B0XCH is executed, the ACC data is transferred to the memory, and data of memory transferred to ACC buffer.

➢ ***B0XCH M,A : Exchange the data between ACC and the BANK 0 memory. The operation doesn't affect PFLAG. (Support R/W register and memory)***

➲ **Example:**

```
        B0XCH         A, WK00        ; Exchange the data between ACC and WK00 defined
                                     ; by user and in BANK 0.
```

MOVC – Read Data from ROM

➲ **Operation:**

Mnemonic	*Description*	*C*	*DC*	*Z*
MOVC	R, A ← ROM [X,Y,Z]	-	-	-

➲ **Description:**

The instruction is to read data of ROM. This is a look-up table application. After defining the X, Y, Z data pointed to the ROM address, to get the ROM data using MOVC instruction. The high byte data is stored in R register and the low byte is stored in ACC buffer.

➢ ***MOVC : Read ROM data and store in R register and ACC. The operation doesn't affect PFLAG.***

➲ **Example: To look up the ROM data located "table_1".**

```
                B0MOV      Y, #TABLE1$M    ; Set look-up table's middle address.
                B0MOV      Z, #TABLE1$L    ; Set look-up table's low address.
                MOVC                       ; Look up data, R = 00H, ACC = 35H
                .                          ;
                INCMS      Z               ; Look up next ROM's data.
                NOP                        ;
                MOVC                       ; Look up data, R = 51H, ACC = 05H.
                .          .               ;
TABLE1:         DW         0035H           ; Define a word (16 bits) data.
                DW         5105H           ; "
                DW         2012H           ; "
                .          .
```

ARITHMETIC INSTRUCTION

ADC – Add with Carry

➲ **Operation:**

Mnemonic		*Description*	*C*	*DC*	*Z*
ADC	A,M	A ← A + M + C, if occur carry, then C=1, else C=0	√	√	√
ADC	M,A	M ← A + M + C, if occur carry, then C=1, else C=0	√	√	√

Remark

1. The "M" is memory including system registers and user defined memory.
2. The "C" is carry flag.

➲ **Description:**

The instruction is to get the sum of ACC, memory and C flag. If the result is zero, the zero flag (Z) will be set as "1"; otherwise the zero flag (Z) is cleared. If the result occurs overflow signal, the carry flag (C) will be set as "1"; otherwise the carry flag (C) is cleared. If there is a carry signal from low nibble of the result, the decimal carry flag (DC) will be set as "1"; otherwise the decimal carry flag (DC) is cleared.

➢ ***ADC A,M : Add ACC, memory and carry flag (C) up, and store the result in ACC buffer. (Support R/W,R register and memory)***

➢ ***ADC M,A : Add ACC, memory and carry flag (C) up, and store the result in the memory. (Support R/W register and memory)***

➲ **Example: The ADC instruction is a useful instruction to do work data adding. There is a word data separated to DAH and DAL. DAH is high byte and DAL is low byte. Add 0x10ff and 0x0103 up and store the sum to DAH and DAL.**

```
MOV       A,#03H      ; Input the low byte (03H) of 0x0103 into DAL
B0MOV     DAL,A
MOV       A,#01H      ; Input the high byte (01H) of 0x0103 into DAH
B0MOV     DAH,A

MOV       A,#0FFH     ; Add low byte data. DAL = 02H
ADD       DAL,A

MOV       A,#10H      ; Add high byte data. DAH = 12H
ADC       DAH,A

                      ; The sum is 0x1202.
```

ADD – Add only

➲ **Operation:**

Mnemonic	*Description*	*C*	*DC*	*Z*
ADD A,M	A ← A + M, if occur carry, then C=1, else C=0	√	√	√
ADD M,A	M ← A + M, if occur carry, then C=1, else C=0	√	√	√
ADD A,I	A ← A + I, if occur carry, then C=1, else C=0	√	√	√

Remark

1. The "M" is memory including system registers and user defined memory.
2. The "C" is carry flag.

➲ **Description:**

The instruction is the add instruction. If the result is zero, the zero flag (Z) will be set as "1"; otherwise the zero flag (Z) is cleared. If the result occurs overflow signal, the carry flag (C) will be set as "1"; otherwise the carry flag (C) is cleared. If there is a carry signal from low nibble of the result, the decimal carry flag (DC) will be set as "1"; otherwise the decimal carry flag (DC) is cleared.

- ***ADD A,M : Add ACC buffer and the memory up, and store the result in ACC buffer. (Support R/W,R register and memory)***
- ***ADD M,A : Add ACC buffer and the memory up, and store the result in the memory. (Support R/W register and memory)***
- ***ADD A,I : Add ACC buffer and the immediate value up, and store the result in ACC buffer.***

➲ **Example:**

```
ADD         A,WK00          ; A = A + WK00

ADD         WK00,A          ; WK00 = A + WK00

ADD         A,#10           ; A = A + 10
```

B0ADD – Add with BANK 0 Memory

➲ **Operation:**

Mnemonic	Description	C	DC	Z
B0ADD M,A	M (bank 0) ← A + M (bank 0), if occur carry, then C=1, else C=0	√	√	√

Remark

1. ***The "M" is memory including system registers and user defined memory.***
2. ***The "C" is carry flag.***

➲ **Description:**

The instruction is the add instruction. The memory must be in BANK0 and the target. If the result is zero, the zero flag (Z) will be set as "1"; otherwise the zero flag (Z) is cleared. If the result occurs overflow signal, the carry flag (C) will be set as "1"; otherwise the carry flag (C) is cleared. If there is a carry signal from low nibble of the result, the decimal carry flag (DC) will be set as "1"; otherwise the decimal carry flag (DC) is cleared.

➢ ***B0ADD M,A : Add ACC buffer and the BANK 0 memory up, and store the result in the memory. (Support R/W register and memory)***

➲ **Example:**

```
        B0ADD           WK00,A                  ; WK00 = A + WK00
```

SBC – Subtract with Carry

➲ **Operation:**

Mnemonic	*Description*	*C*	*DC*	*Z*
SBC A,M	A ← A - M - /C, if occur borrow, then C=0, else C=1	√	√	√
SBC M,A	M ← A - M - /C, if occur borrow, then C=0, else C=1	√	√	√

Remark

1. The "M" is memory including system registers and user defined memory.
2. The "C" is carry flag.

➲ **Description:**

The instruction is to get the difference of ACC, memory and the inverse of C flag. If the result is zero, the zero flag (Z) will be set as "1"; otherwise the zero flag (Z) is cleared. If the result occurs borrow signal, the carry flag (C) will be cleared; otherwise the carry flag (C) is as "1". If there is a borrow signal from low nibble of the result, the decimal carry flag (DC) will be cleared; otherwise the decimal carry flag (DC) is as "1".

➢ ***SBC A,M : Subtract the memory and the inverse of carry flag (C) from ACC buffer, and store the result in ACC buffer. (Support R/W,R register and memory)***

➢ ***SBC M,A : Subtract the memory and the inverse of carry flag (C) from ACC buffer, and store the result in the memory. (Support R/W register and memory)***

➲ **Example: The SBC instruction is a useful instruction to do work data subtract. There is a word data separated to DAH and DAL. DAH is high byte and DAL is low byte. Subtract 0x0103 from 0x10FF and store the result to DAH and DAL.**

```
MOV       A,#03H      ; Input the low byte (03H) of 0x0103 into DAL
B0MOV     DAL,A
MOV       A,#01H      ; Input the high byte (01H) of 0x0103 into DAH
B0MOV     DAH,A

MOV       A,#0FFH     ; Add low byte data. DAL = 0FCH
SUB       DAL,A

MOV       A,#10H      ; Add high byte data. DAH = 0FH
SBC       DAH,A
                      ; The sum is 0x0FFC.
```

SUB – Subtract only

➲ **Operation:**

Mnemonic	*Description*	*C*	*DC*	*Z*
SUB A,M	A ← A - M, if occur borrow, then C=0, else C=1	√	√	√
SUB M,A	M ← A - M, if occur borrow, then C=0, else C=1	√	√	√
SUB A,I	A ← A - I, if occur borrow, then C=0, else C=1	√	√	√

Remark

1. The “M” is memory including system registers and user defined memory.
2. The “C” is carry flag.

➲ **Description:**

The instruction is the subtract instruction. If the result is zero, the zero flag (Z) will be set as “1”; otherwise the zero flag (Z) is cleared. If the result occurs borrow signal, the carry flag (C) will be cleared; otherwise the carry flag (C) is as “1”. If there is a borrow signal from low nibble of the result, the decimal carry flag (DC) will be cleared; otherwise the decimal carry flag (DC) is as “1”.

➢ ***SUB A,M : Subtract the memory from ACC buffer, and store the result in ACC buffer. (Support R/W,R register and memory)***

➢ ***SUB M,A : Subtract the memory from ACC buffer, and store the result in the memory. (Support R/W register and memory)***

➢ ***SUB A,I : Subtract the immediate value from ACC buffer, and store the result in ACC buffer.***

➲ **Example:**

```
SUB     A,WK00     ; A = A - WK00

SUB     WK00,A     ; WK00 = A - WK00

SUB     A,#10      ; A = A - 10
```

DAA –Decimal-Adjust accumulator

➲ **Operation:**

Mnemonic	*Description*	C	DC	Z
DAA	Adjust ACC's data format from HEX to DEC	√	-	-

The instruction is adjust accumulator's data from Hexadecimal to Decimal. The operand of this instruction divided ACC into high and low nibble. Each nibble is done by adding 6 to the original value if the original value is greater than 9; other wise the original value remains unchanged. The result is stored in the accumulator.

For ACC.3~ACC.0 > 9

```
ACC.3~ACC.0 ← (ACC.3~ACC.0)+6
IF (ACC.7~ACC.4+1) > 9
   Then ACC.7~ACC.4 ← (ACC.7~ACC.4)+1+6 ( C flag will set to 1)
Else ACC.7~ACC.4 ← (ACC.7~ACC.4)+1
```

For ACC.3~ACC.0 <=9

```
ACC.3~ACC.0 ← ACC.3~ACC.0
IF ACC.7~ACC.4 > 9
   Then ACC.7~ACC.4 ← (ACC.7~ACC.4)+6 ( C flag will set to 1)
Else ACC.7~ACC.4 ← ACC.7~ACC.4
```

➲ **Example1:**

```
MOV     A,#055H     ; A = 55H
DAA                 ; A = 55H
```

➲ **Example2:**

```
MOV     A,#01FH     ; A = 1FH
DAA                 ; A = 25H
```

MUL – Multiply Unsigned

➲ **Operation:**

Mnemonic	Description	C	DC	Z
MUL A,M	R, A ← A * M, The LB of product stored in Acc and HB stored in R register. ZF affected by Acc.	-	-	√

Remark

The "M" is memory including system registers and user defined memory.

➲ **Description:**

The instruction is multiples 2 8-bit unsigned binary value in ACC and memory. The high byte data of multiple results is stored in R register and the low byte is stored in ACC buffer. If the result is zero, the zero flag (Z) will be set as "1"; otherwise the zero flag (Z) is cleared.

➲ **Example:**

```
            MOV         A, #03H         ; Input the data=03H into DATA_1
            B0MOV       DATA_1,A        ; DATA_1 in bank 0
            MOV         A, #0FFH        ; Input the data=0FFH into ACC
            MUL         A, DATA_1       ; Multiple Data_1 and A
                                        ; R = 02H, A=0FDH, Z_flag=0
   :
            MOV         A, #00H         ; Input the data=00H into DATA_1
            B0MOV       DATA_1,A        ; DATA_1 in bank 0
            MOV         A, #0FFH        ; Input the data=0FFH into ACC
            MUL         A, DATA_1       ; Multiple Data_1 and A
                                        ; R = 00H, A=00H, Z_flag=1
```

LOGIC INSTRUCTION

AND – Logical AND

➲ **Operation:**

Mnemonic	Description	C	DC	Z
AND A,M	A ← A and M	-	-	√
AND M,A	M ← A and M	-	-	√
AND A,I	A ← A and I	-	-	√

Remark

1. The "M" is memory including system registers and user defined memory.
2. The "Z" is zero flag.

➲ **Description:**

The instruction is the logically AND. The logical AND operation results in a "1" bit being stored whenever the corresponding bits in the two operands are both "1". If the result is zero, the zero flag (Z) will be set as "1"; otherwise the zero flag (Z) is cleared.

➢ ***AND A,M : AND the memory and ACC buffer, and store the result in ACC buffer. (Support R/W,R register and memory)***

➢ ***AND M,A : AND the memory and ACC buffer, and store the result in the memory. (Support R/W register and memory)***

➢ ***AND A,I : AND the immediate value and ACC buffer, and store the result in ACC buffer.***

➲ **Example: Read Port 2 value and only keep the low-nibble data.**

```
; Store the result in ACC buffer

              B0MOV         A,P2

              AND           A,#00001111B        ; To clear the high-nibble and compare the low-nibble data.
                                                ; If the P2 = 10101010B.
                                                ; The result is "00001010B".
; or
              B0MOV         A, #00001111B

              AND           A,P2                ; To clear the high-nibble and compare the low-nibble data.
                                                ; If the P2 = 10101010B.
                                                ; The result is "00001010B".

; Store the result in P2 register

              B0MOV         A, #00001111B

              AND           P2,a                ; To clear the high-nibble and compare the low-nibble data.
                                                ; If the P2 = 10101010B.
                                                ; The result is "00001010B".
```

OR – Logical OR

➲ **Operation:**

Mnemonic		Description	C	DC	Z
OR	A,M	A ← A or M	-	-	√
OR	M,A	M ← A or M	-	-	√
OR	A,I	A ← A or I	-	-	√

Remark

1. The "M" is memory including system registers and user defined memory.
2. The "Z" is zero flag.

➲ **Description:**

The instruction is the logically OR. The logical OR operation results in a "1" bit being stored whenever one of the corresponding bits in the two operands is "1". If the result is zero, the zero flag (Z) will be set as "1"; otherwise the zero flag (Z) is cleared.

➢ ***OR A,M : OR the memory and ACC buffer, and store the result in ACC buffer. (Support R/W,R register and memory)***

➢ ***OR M,A : OR the memory and ACC buffer, and store the result in the memory. (Support R/W register and memory)***

➢ ***OR A,I : OR the immediate value and ACC buffer, and store the result in ACC buffer.***

➲ **Example: Read Port 2 value and OR with 0FH.**

```
; Store the result in ACC buffer

              B0MOV     A,P2

              OR        A,#0FH         ; To keep the high-nibble and set all of the low-nibble data
                                       ; as "1".
                                       ; If the P2 = 10101010B.
                                       ; The result is "10101111B".
; or
              B0MOV     A, #0FH

              OR        A,P2           ; To keep the high-nibble and set all of the low-nibble data
                                       ; as "1".
                                       ; If the P2 = 10101010B.
                                       ; The result is "10101111B".

; Store the result in P2 register

              B0MOV     A, #0FH

              OR        P2,a           ; To keep the high-nibble and set all of the low-nibble data
                                       ; as "1".
                                       ; If the P2 = 10101010B.
                                       ; The result is "10101111B".
```

XOR – Logical XOR

➲ **Operation:**

Mnemonic		Description	C	DC	Z
XOR	A,M	A ← A xor M	-	-	√
XOR	M,A	M ← A xor M	-	-	√
XOR	A,I	A ← A xor I	-	-	√

Remark

1. ***The "M" is memory including system registers and user defined memory.***
2. ***The "Z" is zero flag.***

➲ **Description:**

The instruction is the logically XOR. The logical XOR operation results in a "1" bit being stored whenever the corresponding bits in the two operands are the same. If the result is zero, the zero flag (Z) will be set as "1"; otherwise the zero flag (Z) is cleared.

➢ ***XOR A,M : OR the memory and ACC buffer, and store the result in ACC buffer. (Support R/W,R register and memory)***

➢ ***XOR M,A : OR the memory and ACC buffer, and store the result in the memory. (Support R/W register and memory)***

➢ ***XOR A,I : OR the immediate value and ACC buffer, and store the result in ACC buffer.***

➲ **Example: Read Port 2 value and invert it.**

```
; Store the result in ACC buffer

            B0MOV       A,P2

            OR          A,#0FFH         ; To invert P2 register.
                                        ; If the P2 = 10101010B.
                                        ; The result is "01010101B".
; or
            B0MOV       A, #0FFH

            OR          A,P2            ; To invert P2 register.
                                        ; If the P2 = 10101010B.
                                        ; The result is "01010101B".

; Store the result in P2 register

            B0MOV       A, #0FFH

            OR          P2,a            ; To invert P2 register.
                                        ; If the P2 = 10101010B.
                                        ; The result is "01010101B".
```

PROCESS INSTRUCTION

SWAP & SWAPM – Memory high/low Nibble Exchange

➲ **Operation:**

Mnemonic	*Description*	*C*	*DC*	*Z*
SWAP M	A (b3~b0, b7~b4) ←M(b7~b4, b3~b0)	-	-	-
SWAPM M	M(b3~b0, b7~b4) ← M(b7~b4, b3~b0)	-	-	-

Remark

The "M" is memory including system registers and user defined memory.

➲ **Description:**

The instruction's operation is to exchange the high/low byte data of a memory. The operation doesn't affect PFLAG.

➢ ***SWAP M : Exchange the high/low nibble data and store the result in ACC buffer. (Support R/W,R register and memory)***

➢ ***SWAPM M : Exchange the high/low nibble data and store the result in the memory. (Support R/W register and memory)***

➲ **Example: Read P2 value, exchange the high/low nibble data and store in the WK00.**

```
SWAP        P2          ; SWAP P2 register and the result stored in ACC buffer.
B0MOV       WK00,A      ; Save the result in ACC buffer.
```

➲ **Example: Read P2 value, exchange the high/low nibble data and reload the new value into P2 register.**

```
SWAPM       P2          ; SWAP P2 register and the result stored in P2 register.
```

RLC & RLCM – Memory Left Rotation

➲ **Operation:**

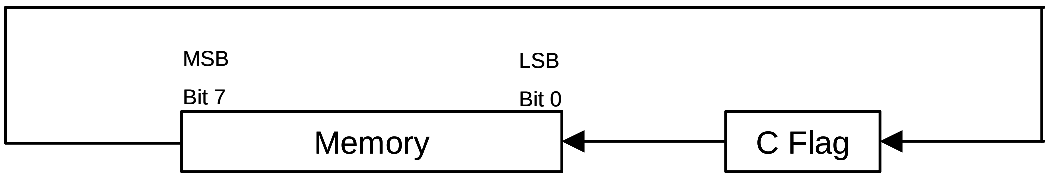
Mnemonic	*Description*	*C*	*DC*	*Z*
RLC M	A ← RLC M	√	-	-
RLCM M	M ← RLC M	√	-	-

Remark

1. ***The "M" is memory including system registers and user defined memory.***
2. ***The "C" is carry flag.***

➲ **Description:**

The instruction rotates the memory from right to left one bit. After RLC /RLCM executed, 8-bit rotate one bit. The LSB is replaced by C flag and the MSB value is moved to C flag position.



- ***RLC M : Rotate the memory to left and store the result in ACC buffer. (Support R/W,R register and memory)***
- ***RLCM M : Rotate the memory to left and store the result in the memory. (Support R/W register and memory)***

➲ **Example: WK00 = 10101010B, change WK00 to 01010101B and store the result in ACC buffer.**

```
B0BSET      FC          ; Set the C flag.

RLC         WK00        ; The ACC buffer = 01010101B
                        ; The C flag = 1 (MSB of WK00).
```

➲ **Example: WK00 = 11110011B, change WK00 to 11001100B and store the result in WK00.**

```
B0BCLR      FC          ; Clear C flag.
RLCM        WK00        ; RLCM one time
                        ; The WK00 = 11100110B
                        ; The C flag = 1 (MSB of WK00).

B0BCLR      FC          ; Clear C flag.
RLCM        WK00        ; RLCM two time
                        ; The WK00 = 11001100B
                        ; The C flag = 1 (MSB of WK00).
```

RRC & RRCM – Memory Right Rotation

➲ **Operation:**

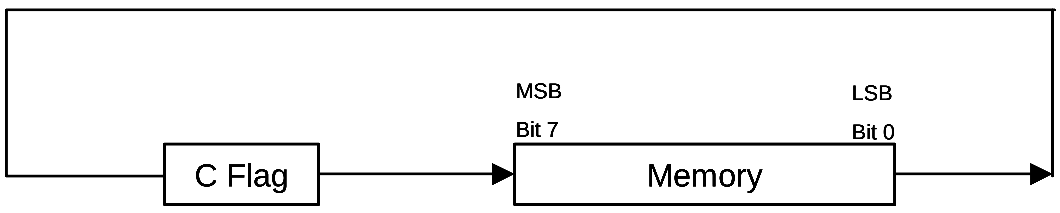
Mnemonic	*Description*	*C*	*DC*	*Z*
RRC M	A ← RRC M	√	-	-
RRCM M	M ← RRC M	√	-	-

Remark

1. ***The "M" is memory including system registers and user defined memory.***
2. ***The "C" is carry flag.***

➲ **Description:**

The instruction rotates the memory from left to right one bit. After RRC /RRCM executed, 8-bit rotate one bit. The MSB is replaced by C flag and the LSB value is moved to C flag position.



➢ ***RRC M : Rotate the memory to right and store the result in ACC buffer. (Support R/W,R register and memory)***

➢ ***RRCM M : Rotate the memory to right and store the result in the memory. (Support R/W register and memory)***

➲ **Example: WK00 = 10101010B, change WK00 to 01010101B and store the result in ACC buffer.**

```
          B0BCLR        FC              ; Clear C flag.

          RRC           WK00            ; The ACC buffer = 01010101B
                                        ; The C flag = 0 (LSB of WK00).
```

➲ **Example: WK00 = 11110011B, change WK00 to 00111100B and store the result in WK00.**

```
          B0BCLR        FC              ; Clear C flag.
          RRCM          WK00            ; RRCM one time
                                        ; The WK00 = 01111001B
                                        ; The C flag = 1 (LSB of WK00).

          B0BCLR        FC              ; Clear C flag.
          RRCM          WK00            ; RRCM two time
                                        ; The WK00 = 00111100B
                                        ; The C flag = 1 (LSB of WK00).
```

CLR – Clear

➲ **Operation:**

Mnemonic	*Description*	*C*	*DC*	*Z*
CLR M	M ← 0	-	-	-

Remark

The "M" is memory including system registers and user defined memory.

➲ **Description:**

The instruction is to clear memory data. The operation is read the memory data, clear and writer back. The operation doesn't affect PFLAG.

➢ ***CLR M : Clear memory value. (Support R/W register and memory)***

➲ **Example: Clear system registers.**

```
CLR     Y       ; Clear Y register
CLR     P0      ; Error. "CLR" doesn't support read only register.
```

BCLR & B0BCLR – Bit Clear

➲ **Operation:**

Mnemonic	Description	C	DC	Z
BCLR M.b	M.b ← 0	-	-	-
B0BCLR M.b	M(bank 0).b ← 0	-	-	-

Remark

The "M" is memory including system registers and user defined memory.

➲ **Description:**

The instruction is to clear a bit of memory. The operation doesn't affect PFLAG.

➢ ***BCLR M.b: Clear the bit called "b" of the memory. (Support R/W register and memory)***

➢ ***B0BCLR M.b: Clear the bit called "b" of the memory. The memory must be in BANK 0. (Support R/W register and memory)***

➲ **Example: Clear carry flag (C).**

```
        B0BCLR        FC
```

➲ **Example: Clear the bit 6 of WK00. WK00 is user defined.**

```
        BCLR          WK00.6             ; Clear the bit 6 of WK00.
```

BSET & B0BSET – Bit Set

➲ **Operation:**

Mnemonic	*Description*	*C*	*DC*	*Z*
BSET M.b	M.b ← 1	-	-	-
B0BSET M.b	M(bank 0).b ← 1	-	-	-

Remark

The "M" is memory including system registers and user defined memory.

➲ **Description:**

The instruction is to set a bit of memory. The operation doesn't affect PFLAG.

➢ ***BSET M.b: Set the bit called "b" of the memory. (Support R/W register and memory)***

➢ ***B0BSET M.b: Set the bit called "b" of the memory. The memory must be in BANK 0. (Support R/W register and memory)***

➲ **Example: Set carry flag (C).**

```
B0BSET      FC
```

➲ **Example: Set the bit 6 of WK00. WK00 is user defined.**

```
BSET        WK00.6          ; Set the bit 6 of WK00.
```

BRANCH INSTRUCTION

CMPRS – Compare

➲ **Operation:**

Mnemonic	*Description*	*C*	*DC*	*Z*
CMPRS A,I	ZF,C ← A - I, If A = I, then skip next instruction	√	-	√
CMPRS A,M	ZF,C ← A - M, If A = M, then skip next instruction	√	-	√

Remark

1. The "M" is memory including system registers and user defined memory.
2. The "C" is carry flag.
3. The "Z" is zero flag.

➲ **Description:**

The instruction compares ACC buffer with the memory or an immediate value. If the result is equal, the program counter will skip next instruction. "CMPRS" is to subtract the memory or an immediate value from ACC buffer. The zero flag and carry flag will be changed by the result. If the result is zero, the zero flag (Z) will be set as "1"; otherwise the zero flag (Z) is cleared. If the result occurs borrow signal, the carry flag (C) will be cleared; otherwise the carry flag (C) is as "1". The decimal carry flag (DC) won't be affect and keeps zero.

➢ ***CMPRS A,I: Compare the ACC buffer with an immediate value.***

➢ ***CMPRS A,M: Compare the ACC buffer with the memory. (Support R/W,R register and memory)***

➲ **Example: Compare ACC buffer with 55H. If ACC is equal to 55H, jump to SUB1, or jump to SUB2.**

```
CMPRS     A,#055H
JMP       SUB2          ; ACC is not 55H.
JMP       SUB1          ; ACC = 55H
```

➲ **Example: Compare the memory called WK00 with 55H. If the memory is equal to 55H, jump to SUB1, or jump to SUB2.**

```
B0MOV     A,#055H

CMPRS     A,WK00
JMP       SUB2          ; WK00 is not 55H.
JMP       SUB1          ; WK00 = 55H
```

INCS & INCMS – Increment and Skip on Zero

➲ **Operation:**

Mnemonic	*Description*	*C*	*DC*	*Z*
INCS M	A ← M + 1, If A = 0, then skip next instruction	-	-	-
INCMS M	M ← M + 1, If M = 0, then skip next instruction	-	-	-

Remark

The "M" is memory including system registers and user defined memory.

➲ **Description:**

The instruction increases the ACC buffer or the memory by one. If the result is zero, the program counter will skip the next instruction. The operation doesn't affect PFLAG.

➢ ***INCS M: Increase the memory value by one and store the result in ACC buffer. (Support R/W register and memory)***

➢ ***INCMS M: Increase the memory value by one and store the result in the memory. (Support R/W register and memory)***

➲ **Example: Increase WK00 until the overflow occurring.**

```
                MOV         A, #0xFD
                MOV         WK00, A
INCSTEST:
                INCS        WK00
                NOP
                MOV         WK00, A
                CMPRS       A, #0x00
                JMP         INCSTEST        ; WK00 is not zero.
                .
```

➲ **Example: Increase Y and Z flags. The Y is high byte and the Z is low byte. Y will increase by one when the Z is overflow.**

```
                INCMS       Z
                JMP EXIT                    ; Z is not zero.

                INCMS       Y               ; Z = 0, Y = Y+1
                NOP
EXIT:
                .
```

DECS & DECMS – Decrement and Skip on Zero

➲ **Operation:**

Mnemonic	Description	C	DC	Z
DECS M	A ← M - 1, If A = 0, then skip next instruction	-	-	-
DECMS M	M ← M - 1, If M = 0, then skip next instruction	-	-	-

Remark

The "M" is memory including system registers and user defined memory.

➲ **Description:**

The instruction decreases the ACC buffer or the memory by one. If the result is zero, the program counter will skip the next instruction. The operation doesn't affect PFLAG.

- ***DECS M: Decrease the memory value by one and store the result in ACC buffer. (Support R/W register and memory)***
- ***DECMS M: Decrease the memory value by one and store the result in the memory. (Support R/W register and memory)***

➲ **Example: Decrease WK00 until equal to zero.**

```
DECSTEST:
                DECMS           WK00
                JMP             DECSTEST            ; Not zero.
                .                                   ; WK00 = 0.
                .
```

BTS0 & B0BTS0 – Bit Test 0

➲ **Operation:**

Mnemonic		*Description*	*C*	*DC*	*Z*
BTS0	M.b	If M.b = 0, then skip next instruction	-	-	-
B0BTS0	M.b	If M(bank 0).b = 0, then skip next instruction	-	-	-

Remark

The "M" is memory including system registers and user defined memory.

➲ **Description:**

The instruction compares the bit of memory with 0. If the result is 0, the program counter will skip the next instruction. The operation doesn't affect PFLAG.

➢ ***BTS0 M.b: Compare the bit of memory with 0. (Support R/W register and memory)***

➢ ***B0BTS0 M.b: Compare the bit of memory with 0. The memory must be in Bank 0. (Support R/W register and memory)***

➲ **Example: Check the carry flag. If C = 0, jump to SUB1, or jump to SUB2.**

```
        B0BTS0          FC
        JMP             SUB2                ; C = 1
        JMP             SUB1                ; C = 0
```

➲ **Example: Check the bit 3 of WK00. If WK00.3 = 0, increase WK00 by one, or exit.**

```
        B0BTS0          WK00.3              ; Check bit 3 of WK00
        JMP             EXIT                ; WK00.3=1
        INCMS           WK00                ; WK00.3=0
        NOP                                 ;

EXIT:
```

BTS1 & B0BTS1 – Bit Test 1

➲ **Operation:**

Mnemonic		*Description*	*C*	*DC*	*Z*
BTS1	M.b	If M.b = 1, then skip next instruction	-	-	-
B0BTS1	M.b	If M(bank 0).b = 1, then skip next instruction	-	-	-

Remark

The "M" is memory including system registers and user defined memory.

➲ **Description:**

The instruction compares the bit of memory with 1. If the result is 1, the program counter will skip the next instruction. The operation doesn't affect PFLAG.

➢ ***BTS1 M.b: Compare the bit of memory with 1. (Support R/W register and memory)***

➢ ***B0BTS1 M.b: Compare the bit of memory with 1. The memory must be in Bank 0. (Support R/W register and memory)***

➲ **Example: Check the carry flag. If C = 1, jump to SUB1, or jump to SUB2.**

```
        B0BTS1      FC
        JMP         SUB2          ; C = 0
        JMP         SUB1          ; C = 1
```

➲ **Example: Check the bit 3 of WK00. If WK00.3 = 1, increase WK00 by one, or exit.**

```
        B0BTS1      WK00.3        ; Check bit 3 of WK00
        JMP         EXIT          ; WK00.3=0
        INCMS       WK00          ; WK00.3=1
        NOP                       ;

EXIT:
```

JMP – Jump

➲ **Operation:**

Mnemonic	Description	C	DC	Z
JMP d	PC15/14 ← RomPages1/0, PC13~PC0 ← d	-	-	-

Remark

The "d" is the label name of the destination address.

➲ **Description:**

"JMP" is a unconditional branch to indicate address by replacing the contents of the program counter (PC) with the destination address. The destination address can be anywhere in the program memory (ROM). The operation doesn't affect PFLAG.

➢ ***JMP d: Program jumps to "d" address.***

➲ **Example: Jump to SUB1.**

```
                JMP             SUB1
                .
SUB1:
                .
```

➲ **Example: Using program counter and JMP instruction can be a jump table function. Program can execute different subroutines by the program counter.**

```
                MOV             A,WK00          ; Limit WK00 from 0 to 7.
                AND             A,#00000111B
                MOV             WK00,A

                B0ADD           PCL,A
                JMP             SUB0            ; WK00=0
                JMP             SUB1            ; WK00=1
                JMP             SUB2            ; WK00=2
                JMP             SUB3            ; WK00=3
                JMP             SUB4            ; WK00=4
                JMP             SUB5            ; WK00=5
                JMP             SUB6            ; WK00=6
                JMP             SUB7            ; WK00=7
                .
                .
                .
```

CALL – Call Procedure

➲ **Operation:**

Mnemonic	Description	C	DC	Z
CALL d	Stack ← PC15~PC0, PC15/14 ← RomPages1/0, PC13~PC0 ← d	-	-	-

Remark

The "d" is the label name of the destination address.

➲ **Description:**

"CALL" calls a subroutine in the program memory (ROM). When the instruction is executing, the program counter is stored in stack buffer and the stack point decrease by one. The new address of the subroutine label is loaded in program counter and executed. The subroutine can begin anywhere in all program memory (ROM). The operation doesn't affect PFLAG.

➢ ***CALL d: Program jumps to "d" address.***

➲ **Example: The SUB1 address is 0x0213. The CALL command address is 0x0030. The stack point is 0FH. After executing CALL, the stack buffer is 0x0031,stack point is 0EH (decrease by one) and the program counter is 0x0213.**

```
        ORG         0X0030
        CALL        SUB1            ; Address = 0x0030
        .                           ; Address = 0x0031. Program return address.
        .

        ORG         0X0213
SUB1:                               ; Address = 0x0213
        .
        .
        RET
```

MISCELLANEOUS INSTRUCTION

RET – Return from Call Procedure

➲ **Operation:**

Mnemonic	*Description*	*C*	*DC*	*Z*
RET	PC ← Stack	-	-	-

➲ **Description:**

"RET" returns to the last ROM address after finishing the call subroutine. When the instruction is executing, the program counter is re-loaded from stack buffer and the stack point increase by one. The program returns to last stack and continues to execute. The subroutine can end anywhere in all program memory (ROM). The operation doesn't affect PFLAG.

➢ ***RET: Return to the last stack level.***

➲ **Example: The SUB1 address is 0x0213. The CALL command address is 0x0030. The stack point is 0FH. After executing RET, the stack buffer is load to program counter, stack point is 0FH (increase by one) and the program continues running from 0x0031.**

```
        ORG         0X0030
        CALL        SUB1          ; Address = 0x0030.
        .                         ; Address = 0x0031. Program return address.
        .

        ORG         0X0213
SUB1:                             ; Address = 0x0213
        .
        .
        RET                       ; Return to 0x0031
```

RETI – Return from Interrupt

➲ **Operation:**

Mnemonic	*Description*	*C*	*DC*	*Z*
RETI	PC ← Stack, and to enable global interrupt	-	-	-

➲ **Description:**

When any interrupt occurs, the program counter content is stored into stack buffer and the stack point increases by one. The global interrupt flag disables, the program jumps to interrupt vector (ORG 8) and executes the interrupt service routine. The RETI is the end of interrupt service routine. It loads stack buffer content to program counter and enable the global interrupt flag. And program exit the interrupt service routine. The operation doesn't affect PFLAG.

➢ ***RETI: Return from interrupt and enable global interrupt flag.***

➲ **Example:**

```
                ORG             8                       ; Interrupt vector.
                B0XCH           A,ACCBUF
                B0MOV           A,PFLAG
                B0MOV           PFLAGBUF,A
                .
                .
                .
                B0MOV           A,PFLAGBUF
                B0MOV           PFLAG,A
                B0XCH           A,ACCBUF
                RETI                                    ; Exit interrupt service routine.


MAIN:
                .
                .                                       ; An interrupt occurs. Program jumps to interrupt vector.
                .                                       ; Return from interrupt vector.
                .
```

NOP – No Operation

➲ **Operation:**

Mnemonic	*Description*	*C*	*DC*	*Z*
NOP	No operation	-	-	-

➲ **Description:**

"NOP" is no operation. It is typically used from timing delay. One "NOP" instruction is one instruction cycle (Fcpu). The operation doesn't affect PFLAG.

➢ ***NOP : No operation.***

➲ **Example: A 1024 instruction cycles delay time. If the oscillator is 4MHz, the delay is 1024u seconds.**

```
DLY1024U:
            MOV         A,#0            ; Clear ACC buffer

DLY1024U_10:
            ADD         A,#01H          ; Increase ACC by one
            B0BTS1      FZ              ; Check ACC overflow.
            JMP         DLY1024U_10     ; No overflow, continues to increase ACC.
            .                           ; Exit the delay routine.
            .
```

PUSH – Save Some System Registers for SN8P1XXX and SN8P270XA

`

➲ **Operation:**

Mnemonic	*Description*	*C*	*DC*	*Z*
PUSH	To push working registers (080H~087H) into buffers	-	-	-

➲ **Description:**

There are two kinds of PUSH instruction. In SN8P1XXX series and SN8P270XA (SN8P2704A, SN8P2705A, SN8P2706A, SN8P2707A and SN8P2708A), the push instruction save some system registers (80H ~ 87H) into buffers. Please refer to datasheet for detailed information.

➢ ***PUSH: Save some system registers (80H ~ 87H) into buffers (doesn't occupy stack buffer).***

➲ **Example: Save 80H ~ 87H system registers after enter interrupt service routine**

```
INT_SERVICE:

            B0XCH       A, ACCBUF        ; B0XCH doesn't change C, Z flag
            PUSH                         ; Save 80H ~ 87H system register
            .           .
            .           .
EXIT_INT:
            POP                          ; Pop 80H ~ 87H system register
            B0XCH       A, ACCBUF        ; Restore ACC value.

            RETI                         ; Exit interrupt vector
```

PUSH – Save A and PFLAG for SN8P260X

`

➲ Operation:

Mnemonic	Description	C	DC	Z
PUSH	To push ACC and PFLAG (except NT0, NPD bit) into buffers.	-	-	-

➲ Description:

There are two kinds of PUSH instruction. In SN8P260X series (SN8P2602A, SN8P2604, SN8P2606 and SN8P2608), the push instruction save Accumulator and PFLAG register (except NT0, NPD bits) into buffers. Please refer to datasheet for detailed information.

➢ ***PUSH: Save Accumulator and PFLAG (except NT0, NPD bits) into buffers (doesn't occupy stack buffer).***

➲ Example: Save A and PFLAH after enter interrupt service routine

```
                ORG             8                   ; Interrupt vector
                JMP             INT_SERVICE
INT_SERVICE:

                PUSH                                ; Save ACC and PFLAG to buffers.

                .               .
                .               .
EXIT_INT:
                POP                                 ; Load ACC and PFLAG from buffers.


                RETI                                ; Exit interrupt vector
```

POP – Restore Some System Registers for SN8P1XXX and SN8P270XA

`

➲ Operation:

Mnemonic	Description	C	DC	Z
POP	To pop working registers (080H~087H) from buffers	√	√	√

➲ Description:

There are two kinds of POP instruction. In SN8P1XXX series and SN8P270XA (SN8P2704A, SN8P2705A, SN8P2706A, SN8P2707A and SN8P2708A), the pop instruction restores some system registers (80H ~ 87H) from buffers. Please refer to datasheet for detailed information.

➢ ***POP: Restore some system registers (80H ~ 87H) from buffers (not from stack buffer).***

➲ Example: Restore 80H ~ 87H system registers before exit interrupt service routine

```
INT_SERVICE:

                B0XCH           A, ACCBUF           ; B0XCH doesn't change C, Z flag
                PUSH                                ; Save 80H ~ 87H system registers
                .               .
                .               .
EXIT_INT:
                POP                                 ; Restore 80H ~ 87H system registers
                B0XCH           A, ACCBUF           ; Restore ACC value.

                RETI                                ; Exit interrupt vector
```

POP – Restore A and PFLAG for SN8P260X

`

➲ **Operation:**

Mnemonic	*Description*	*C*	*DC*	*Z*
POP	To pop ACC and PFLAG (except NT0, NPD bit) from buffers.	√	√	√

➲ **Description:**

There are two kinds of POP instruction. In SN8P260X series (SN8P2602A, SN8P2604, SN8P2606 and SN8P2608), the pop instruction restores Accumulator and PFLAG register (except NT0, NPD bits) from buffers. Please refer to datasheet for detailed information.

➢ ***POP: Restore Accumulator and PFLAG (except NT0, NPD bits) from buffers (not from stack buffer).***

➲ **Example: Restore A and PFLAH before exit interrupt service routine**

```
                ORG             8               ; Interrupt vector
                JMP             INT_SERVICE
INT_SERVICE:

                PUSH                            ; Save ACC and PFLAG to buffers.

                .               .
                .               .
EXIT_INT:
                POP                             ; Load ACC and PFLAG from buffers.


                RETI                            ; Exit interrupt vector
```

Main Office:
Address: 9F, NO. 8, Hsien Cheng 5th St, Chupei City, Hsinchu, Taiwan R.O.C.
Tel: 886-3-551 0520
Fax: 886-3-551 0523
Taipei Office:
Address: 15F-2, NO. 171, Song Ted Road, Taipei, Taiwan R.O.C.
Tel: 886-2-2759 1980
Fax: 886-2-2759 8180
Hong Kong Office:
Address: Flat 3 9/F Energy Plaza 92 Granville Road, Tsimshatsui East Kowloon.
Tel: 852-2723 8086
Fax: 852-2723 9179
Technical Support by Email:
Sn8fae@sonix.com.tw