CHAPTER 8 INSTRUCTION DESCRIPTIONS

This chapter describes the instructions of the 78K/II series products. Instructions with different operands are grouped by mnemonic.

The basic organization of the instruction descriptions is shown on the next page.

See **Chapter 7** for the number of bytes and operation codes of the instructions.

The same instructions are used on all 78K/II series products.

Description Example

Mnemonic

Full name

MOV

Move

Byte Data Transfer

Meaning of instruction

[Instruction format] **MOV dst, src:** Shows the basic coding format of the instruction

[Operation] **dst <- src:** Shows the instruction operation using symbols.

[Operands] Shows the operands which can be specified with this instruction. See **Chapter 7** for the symbols used for the various operands.

Mnemonic	Operands
MOV	r, #byte
	saddr, #byte
	A, saddr
	saddr, A
	A, mem

Mnemonic	Operands
MOV	A, &mem
	A, !addr16
	A, &!addr16
	PSW, A
	A, PSW

[Flags] Shows the operation of flags which are changed by execution of the instruction. The symbols used for flag operations are shown below.

Z	AC	CY

Legend

Symbol	Meaning
Blank	No change
0	Cleared to 0
1	Set to 1
x	Set or cleared depending on result
R	Previously saved value is restored

[Description] Describes the operation of the instruction in detail.

- Transfers the contents of the source operand (src) specified by the 2nd operand to the destination operand (dst) specified by the 1st operand.

[Coding example]

MOV A, #4DH ; Transfers 4DH to the A register

8.1 8-BIT DATA TRANSFER INSTRUCTIONS

8-bit data transfer instructions are as follows:

MOV	...	144
XCH	...	145

MOV — Move Byte Data Transfer

[Instruction format] **MOV dst, src**

[Operation] **dst <- src**

[Operands]

Mnemonic	Operands (dst, src)	Mnemonic	Operands (dst, src)
MOV	r, #byte	**MOV**	A, &mem
	saddr, #byte		mem, A
	sfr, #byte		&mem, A
	r, r'		A, !addr16
	A, r		A, &!addr16
	A, saddr		!addr16, A
	saddr, A		&!addr16, A
	saddr, saddr'		PSW, #byte
	A, sfr		PSW, A
	sfr, A		A, PSW
	A, mem		

[Flags]

In case of PSW, #byte and PSW, A operands

Z	AC	CY
x	x	x

Other than cases at left

Z	AC	CY

[Description]

- Transfers the contents of the source operand (src) specified by the 2nd operand to the destination operand (dst) specified by the 1st operand.
- No interrupts or macro services are acknowledged between MOV PSW, #byte instruction, MOV PSW, A instruction and the next instruction.

[Coding example]

MOV A, #4DH ; Transfer 4DH to the A register

XCH

Exchange
Byte Data Exchange

[Instruction format] **XCH dst, src**

[Operation] **dst <-> src**

[Operands]

Mnemonic	Operands (dst, src)
XCH	A, r
	r, r'
	A, mem
	A, &mem
	A, saddr
	A, sfr
	saddr, saddr'

[Flags]

Z	AC	CY

[Description]

- Exchanges the contents of the 1st operand with contents of the 2nd operand.

[Coding example]

XCH A, OFEBCH ; Exchanges the contents of register A with the contents of address OFEBCH.

8.2 16-BIT DATA TRANSFER INSTRUCTIONS

16-bit data transfer instructions are as follows:

MOVW ... 147

MOVW — Move Word / Word Data Transfer

[Instruction format] **MOVW dst, src**

[Operation] **dst <- drc**

[Operands]

Mnemonic	Operands (dst, src)
MOVW	rp, #word
	saddrp, #word
	sfrp, #word
	rp, rp'
	AX, saddrp
	saddrp, AX
	AX, sfrp
	sfrp, AX
	AX, mem1
	AX, &mem1
	mem1, AX[Note]
	&mem1, AX[Note]

Note Cannot be used for the EEPROM area of the μPD78244 sub-series.

[Flags]

Z	AC	CY

[Description]

- Transfers the contents of the source operand (src) specified by the 2nd operand to the destination operand (dst) specified by the 1st operand.

[Coding example]

MOVW AX, [HL] ; Transfers the memory contents indicated by the HL register to the AX register.

8.3 8-BIT OPERATION INSTRUCTIONS

8-bit operation instructions are as follows:

ADD

Add
Byte Data Addition

[Instruction format] **ADD dst, src**

[Operation] **dst, CY <- dst + src**

[Operands]

Mnemonic	Operands (dst, src)
ADD	A, #byte
	saddr, #byte
	sfr, #byte
	r, r'
	A, saddr
	A, sfr
	saddr, saddr'
	A, mem
	A, &mem

[Flags]

Z	AC	CY
x	x	x

[Description]

- Adds the source operand (src) specified by the 2nd operand to the destination operand (dst) specified by the 1st operand, and stores the result in the CY flag and the destination operand (dst).
- If dst is 0 as a result of the addition the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a carry is generated out of bit 7 as a result of the addition the CY flag is set (1), otherwise the CY flag is cleared (0).
- If a carry is generated out of bit 3 into bit 4 as a result of the addition the AC flag is set (1), otherwise the AC flag is cleared (0).

[Coding example]

ADD CR11, #56H ; Adds 56H to the CR11 register and stores the result in the CR11 register.

ADDC

Add with Carry
Byte Data Addition including Carry

[Instruction format] **ADDC dst, src**

[Operation] **dst, CY <- dst + src + CY**

[Operands]

Mnemonic	Operands (dst, src)
ADDC	A, #byte
	saddr, #byte
	sfr, #byte
	r, r'
	A, saddr
	A, sfr
	saddr, saddr'
	A, mem
	A, &mem

[Flags]

Z	AC	CY
x	x	x

[Description]

- Adds the source operand (src) specified by the 2nd operand and the CY flag to the destination operand (dst) specified by the 1st operand, and stores the result in the destination operand (dst) and the CY flag. The CY flag is added to the LSB.
  This instruction is mainly used in multiple-byte additions.
- If dst is 0 as a result of the addition the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a carry is generated out of bit 7 as a result of the addition the CY flag is set (1), otherwise the CY flag is cleared (0).
- If a carry is generated out of bit 3 into bit 4 as a result of the addition the AC flag is set (1), otherwise the AC flag is cleared (0).

[Coding example]

ADDC A, 1234H[B] ; Adds the A register, the contents of address (1234H + (B register)) and the CY flag, and stores the result in the A register.

SUB

Subtract
Byte Data Subtraction

[Instruction format] **SUB dst, src**

[Operation] **dst, CY <- dst – src**

[Operands]

Mnemonic	Operands (dst, src)
SUB	A, #byte
	saddr, #byte
	sfr, #byte
	r, r'
	A, saddr
	A, sfr
	saddr, saddr'
	A, mem
	A, &mem

[Flags]

Z	AC	CY
x	x	x

[Description]

- Subtracts the source operand (src) specified by the 2nd operand from the destination operand (dst) specified by the 1st operand, and stores the result in the destination operand (dst) and the CY flag. The destination operand can be cleared to 0 by making the source operand (src) and the destination operand (dst) the same.
- If dst is 0 as a result of the subtraction the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a borrow is generated in bit 7 as a result of the subtraction the CY flag is set (1), otherwise the CY flag is cleared (0).
- If a borrow is generated out of bit 4 into bit 3 as a result of the subtraction the AC flag is set (1), otherwise the AC flag is cleared (0).

[Coding example]

SUB D, L ; Subtracts the L register from the D register and stores the result in the D register.

SUBC

Subtract with Carry
Byte Data Subtraction including Carry

[Instruction format] **SUBC dst, src**

[Operation] **dst, CY <- dst – src – CY**

[Operands]

Mnemonic	Operands (dst, src)
SUBC	A, #byte
	saddr, #byte
	sfr, #byte
	r, r'
	A, saddr
	A, sfr
	saddr, saddr'
	A, mem
	A, &mem

[Flags]

Z	AC	CY
x	x	x

[Description]

- Subtracts the source operand (src) specified by the 2nd operand and the CY flag from the destination operand (dst) specified by the 1st operand, and stores the result in the destination operand (dst).
  The CY flag is subtracted from the LSB.
  This instruction is mainly used in multiple-byte subtractions.
- If dst is 0 as a result of the subtraction the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a borrow is generated in bit 7 as a result of the subtraction the CY flag is set (1), otherwise the CY flag is cleared (0).
- If a borrow is generated out of bit 4 into bit 3 as a result of the subtraction the AC flag is set (1), otherwise the AC flag is cleared (0).

[Coding example]

SUBC A, [DE+] ; Subtracts the contents of (DE register) address and the CY flag from the A register and stores the result in the A register. (The DE register is incremented after the operation.)

AND

And
Byte Data Logical Product

[Instruction format] **AND dst, src**

[Operation] **dst <- dst ∧ src**

[Operands]

Mnemonic	Operands (dst, src)
AND	A, #byte
	saddr, #byte
	sfr, #byte
	r, r'
	A, saddr
	A, sfr
	saddr, saddr'
	A, mem
	A, &mem

[Flags]

Z	AC	CY
x		

[Description]

- Obtains the bit-by-bit logical product of the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand, and stores the result in the destination operand (dst).
- If all bits are 0 as a result of obtaining the logical product, the Z flag is set (1), otherwise the Z flag is cleared (0).

[Coding example]

AND 0FEBAH, #11011100B ;Obtains the bit-by-bit logical product of the contents of 0FEBAH and 11011100B, and stores the result in 0FEBAH.

OR

Or
Byte Data Logical Sum

[Instruction format] **OR dst, src**

[Operation] **dst <- dst ∨ src**

[Operands]

Mnemonic	Operands (dst, src)
OR	A, #byte
	saddr, #byte
	sfr, #byte
	r, r'
	A, saddr
	A, sfr
	saddr, saddr'
	A, mem
	A, &mem

[Flags]

Z	AC	CY
x		

[Description]

- Obtains the bit-by-bit logical sum of the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand, and stores the result in the destination operand (dst).
- If all bits are 0 as a result of obtaining the logical sum, the Z flag is set (1), otherwise the Z flag is cleared (0).

[Coding example]

OR A, 0FE98H ; Obtains the bit-by-bit logical sum of the A register and 0FE98H, and stores the result in the A register.

XOR

Exclusive Or
Byte Data Exclusive Logical Sum

[Instruction format] **XOR dst, src**

[Operation] **dst <- dst ∀ src**

[Operands]

Mnemonic	Operands (dst, src)
XOR	A, #byte
	saddr, #byte
	sfr, #byte
	r, r'
	A, saddr
	A, sfr
	saddr, saddr'
	A, mem
	A, &mem

[Flags]

Z	AC	CY
x		

[Description]

- Obtains the bit-by-bit exclusive logical sum of the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand, and stores the result in the destination operand (dst).
  The logical negation of all bits of the destination operand (dst) can be obtained by selecting #0FFH as the source operand (src) of this instruction.
- If all bits are 0 as a result of obtaining the exclusive logical sum, the Z flag is set (1), otherwise the Z flag is cleared (0).

[Coding example]

XOR A, P2 ; Obtains the bit-by-bit exclusive logical sum of the A register and the P2 register, and stores the result in the A register.

CMP

Compare
Byte Data Comparison

[Instruction format] **CMP dst, src**

[Operation] **dst – src**

[Operands]

Mnemonic	Operands (dst, src)
CMP	A, #byte
	saddr, #byte
	sfr, #byte
	r, r'
	A, saddr
	A, sfr
	saddr, saddr'
	A, mem
	A, &mem

[Flags]

Z	AC	CY
x	x	x

[Description]

- Subtracts the source operand (src) specified by the 2nd operand from the destination operand (dst) specified by the 1st operand.
  The result of the subtraction is not stored anywhere; only the Z, AC and CY flags are changed.
- If the result of the subtraction is 0 the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a borrow is generated in bit 7 as a result of the subtraction the CY flag is set (1), otherwise the CY flag is cleared (0).
- If a borrow is generated out of bit 4 into bit 3 as a result of the subtraction the AC flag is set (1), otherwise the AC flag is cleared (0).

[Coding example]

CMP 0FE38H, 0FED0H ; Subtracts the contents of address 0FED0H from the contents of address 0FE38H, and performs flag modification only (comparison of address 0FE38H contents and address 0FED0H contents).

8.4 16-BIT OPERATION INSTRUCTIONS

16-bit operation instructions are as follows:

ADDW

Add Word
Word Data Addition

[Instruction format] **ADDW dst, src**

[Operation] **dst, CY <- dst + src**

[Operands]

Mnemonic	Operands (dst, src)
ADDW	AX, #word
	AX, rp
	AX, saddrp
	AX, sfrp

[Flags]

Z	AC	CY
x	x	x

[Description]

- Adds the source operand (src) specified by the 2nd operand to the destination operand (dst) specified by the 1st operand, and stores the result in the destination operand (dst).
- If dst is 0 as a result of the addition the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a carry is generated out of bit 15 as a result of the addition the CY flag is set (1), otherwise the CY flag is cleared (0).
- The AC flag is undefined as a result of the addition.

[Coding example]

ADDW AX, #0AB0DH ; Adds 0ABCDH to the AX register and stores the result in the AX register.

SUBW

Subtract Word
Word Data Subtraction

[Instruction format] **SUBW dst, src**

[Operation] **dst, CY <- dst – src**

[Operands]

Mnemonic	Operands (dst, src)
SUBW	AX, #word
	AX, rp
	AX, saddrp
	AX, sfrp

[Flags]

Z	AC	CY
x	x	x

[Description]

- Subtracts the source operand (src) specified by the 2nd operand from the destination operand (dst) specified by the 1st operand, and stores the result in the destination operand (dst) and the CY flag.
  The destination operand can be cleared to 0 by making the source operand (src) and the destination operand (dst) the same.
- If dst is 0 as a result of the subtraction the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a borrow is generated in bit 15 as a result of the subtraction the CY flag is set (1), otherwise the CY flag is cleared (0).
- The AC flag is undefined as a result of the subtraction.

[Coding example]

SUBW AX, CR01 ; Subtracts the contents of the CR01 register from the contents of the AX register, and stores the result in the AX register.

CMPW

Compare Word
Word Data Comparison

[Instruction format] **CMPW dst, src**

[Operation] **dst – src**

[Operands]

Mnemonic	Operands (dst, src)
CMPW	AX, #word
	AX, rp
	AX, saddrp
	AX, sfrp

[Flags]

Z	AC	CY
x	x	x

[Description]

- Subtracts the source operand (src) specified by the 2nd operand from the destination operand (dst) specified by the 1st operand.
  The result of the subtraction is not stored anywhere; only the Z, AC and CY flags are changed.
- If a result of the subtraction is 0 the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a borrow is generated in bit 15 as a result of the subtraction the CY flag is set (1), otherwise the CY flag is cleared (0).
- The AC flag is undefined as a result of the subtraction.

[Coding example]

CMPW AX, 0FE43H ; Subtracts the word data in address 0FE43H from the AX register, and performs flag modification only (comparison of the AX register and address 0FE43H word data).

8.5 MULTIPLICATION/DIVISION INSTRUCTIONS

Multiplication/division instructions are as follows:

MULU

Multiply Unsigned
Unsigned Data Multiplication

[Instruction format] **MULU src**

[Operation] **AX <- AX x src**

[Operands]

Mnemonic	Operands (src)
MULU	r

[Flags]

Z	AC	CY

[Description]

- Multiplies the A register contents by the source operand (src) data as unsigned data, and stores the result in the AX register.

[Coding example]

MULU H ; Multiplies the A register contents by the H register contents and stores the result in the AX register.

DIVUW

Divide Unsigned Word
Unsigned Word Data Division

[Instruction format] **DIVUW dst**

[Operation] **AX (quotient), dst (remainder) <- AX/dst**

[Operands]

Mnemonic	Operands (dst)
DIVUW	r

[Flags]

Z	AC	CY

[Description]

- Divides the AX register contents by the destination operand (dst) contents, and stores the quotient in the AX register and the remainder in the destination operand (dst).
  The division treats the contents of the AX register and the destination operand (dst) as unsigned data.
- Dividing the contents by 0 (dst = 0) results in:
  AX (quotient) = FFFFH
  dst (remainder) = Original value of the X register

[Coding example]

DIVUW E ; Divides the AX register contents by the E register contents, and stores the quotient in the AX register and the remainder in the E register.

8.6 INCREMENT/DECREMENT INSTRUCTIONS

Increment/decrement instructions are as follows:

INC

Increment
Byte Data Increment

[Instruction format] **INC dst**

[Operation] **dst <- dst + 1**

[Operands]

Mnemonic	Operands (dst)
INC	r
	saddr

[Flags]

Z	AC	CY
x	x	

[Description]
- Increments the contents of the destination operand (dst) by 1.
- If the result of the increment is 0 the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a carry is generated out of bit 3 into bit 4 as a result of the increment the AC flag is set (1), otherwise the AC flag is cleared (0).
- Since this instruction is often used to increment a counter for repeated processing or the offset register in indexed addressing, the contents of the CY flag are not changed (in order to retain the CY flag contents in a multiple-byte operation).

[Coding example]
INC B ; Increments the B register.

DEC

Decrement
Byte Data Decrement

[Instruction format] **DEC dst**

[Operation] **dst <- dst – 1**

[Operands]

Mnemonic	Operands (dst)
DEC	r
	saddr

[Flags]

Z	AC	CY
x	x	

[Description]

- Decrements the contents of the destination operand (dst) by 1.
- If the result of the decrement is 0 the Z flag is set (1), otherwise the Z flag is cleared (0).
- If a carry is generated out of bit 4 into bit 3 as a result of the decrement the AC flag is set (1), otherwise the AC flag is cleared (0).
- Since this instruction is often used to decrement a counter for repeated processing or the offset register in indexed addressing, the contents of the CY flag are not changed (in order to retain the CY flag contents in a multiple-byte operation).
- If it is not wished to change the AC and CY flags when dst is the B register, C register or saddr, the DBNZ instruction can be used.

[Coding example]

DEC 0FE92H ; Decrements the contents of address 0FE92H.

INCW

Increment Word
Word Data Increment

[Instruction format] **INCW dst**

[Operation] **dst <- dst + 1**

[Operands]

Mnemonic	Operands (dst)
INCW	rp

[Flags]

Z	AC	CY

[Description]

- Increments the contents of the destination operand (dst) by 1.
- Since this instruction is often used to increment the register (pointer) in addressing which uses a register, the contents of the Z, AC and CY flags are not changed.

[Coding example]

INCW HL ; Increments the HL register.

DECW

Decrement Word
Word Data Decrement

[Instruction format] **DECW dst**

[Operation] **dst <- dst – 1**

[Operands]

Mnemonic	Operands (dst)
DECW	rp

[Flags]

Z	AC	CY

[Description]

- Decrements the contents of the destination operand (dst) by 1.
- Since this instruction is often used to decrement the register (pointer) in addressing which uses a register, the contents of the Z, AC and CY flags are not changed.

[Coding example]

DECW DE ; Decrements the DE register.

8.7 SHIFT/ROTATE INSTRUCTIONS

Shift/rotate instructions are as follows:

ROR

Rotate Right
Byte Data Right Rotation

[Instruction format] **ROR dst, cnt**

[Operation] **(CY, dst_7 <- dst_0, dst_{m-1} <- dst_m) x cnt times cnt = 0 to 7**

[Operands]

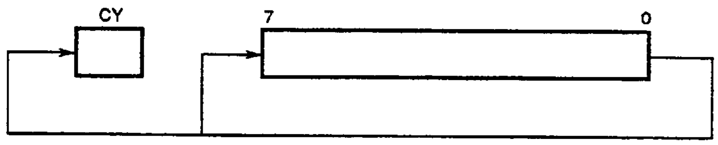
Mnemonic	Operands (dst, cnt)
ROR	r, n

[Flags]

Z	AC	CY
		x

[Description]

- Rotates to the right the contents of the destination operand (dst) specified by the 1st operand cnt times as specified by the 2nd operand.
- The contents of the LSB (bit 0) are rotated into the MSB (bit 7) and at the same time transferred to the CY flag.
- ★ If the 2nd operand (cnt) is 0, no processing is performed. (Z, AC, and CY flags also do not change.)



[Coding example]

ROR C, 4 ; Rotates the contents of the C register 4 bits to the right.

ROL

Rotate Left
Byte Data Left Rotation

[Instruction format] **ROL dst, cnt**

[Operation] **(CY, dst_0 <- dst_7, dst_{m+1} <- dst_m) x cnt times cnt = 0 to 7**

[Operands]

Mnemonic	Operands (dst, cnt)
ROL	r, n

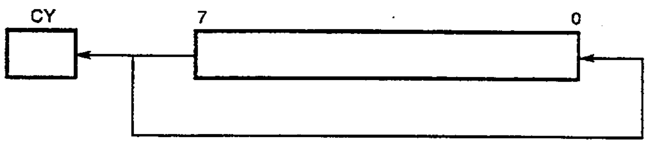
[Flags]

Z	AC	CY
		x

[Description]

- Rotates to the left the contents of the destination operand (dst) specified by the 1st operand cnt times as specified by the 2nd operand.
- The contents of the MSB (bit 7) are rotated into the LSB (bit 0) and at the same time transferred to the CY flag.
- If the 2nd operand (cnt) is 0, no processing is performed.

*



[Coding example]

ROL L, 2 ; Rotates the contents of the L register 2 bits to the left.

RORC

Rotate Right with Carry
Byte Data Right Rotation including Carry

[Instruction format] **RORC dst, cnt**

[Operation] **($CY \leftarrow dst_0$, $dst_7 \leftarrow CY$, $dst_{m-1} \leftarrow dst_m$) x cnt times cnt = 0 to 7**

[Operands]

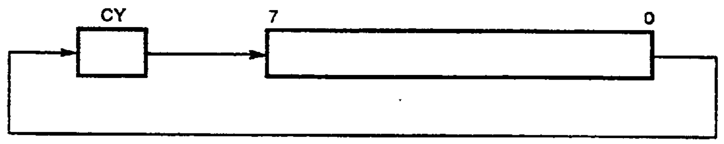
Mnemonic	Operands (dst, cnt)
RORC	r, n

[Flags]

Z	AC	CY
		x

[Description]

- Rotates to the right the contents of the destination operand (dst) specified by the 1st operand, including the CY flag, cnt times as specified by the 2nd operand.
- * If the 2nd operand (cnt) is 0, no processing is performed. (Z, AC, and CY flags also do not change.)



[Coding example]

RORC B, 1 ; Rotates the contents of the B register 1 bit to the right, including the CY flag.

ROLC

Rotate Left with Carry
Byte Data Left Rotation including Carry

[Instruction format] **ROLC dst, cnt**

[Operation] **$(CY \leftarrow dst_7,\ dst_0 \leftarrow CY,\ dst_{m+1} \leftarrow dst_m) \times cnt$ times cnt = 0 to 7**

[Operands]

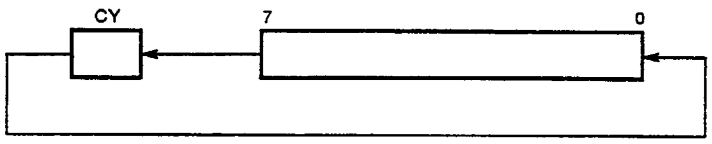
Mnemonic	Operands (dst, cnt)
ROLC	r, n

[Flags]

Z	AC	CY
		x

[Description]

- Rotates to the left the contents of the destination operand (dst) specified by the 1st operand, including the CY flag, cnt times as specified by the 2nd operand.
- If the 2nd operand (cnt) is 0, no processing is performed. (Z, AC, and CY flags also do not change.) ★
- To perform a one-bit left rotation, execution time can be reduced by using ADDC r, r.



[Coding example]

ROLC A, 3 ; Rotates the contents of the A register 3 bits to the left, including the CY flag.

SHR

Shift Right (Logical)
Byte Data Logical Right Shift

[Instruction format] **SHR dst, cnt**

[Operation] **($CY \leftarrow dst_0$, $dst_7 \leftarrow 0$, $dst_{m-1} \leftarrow dst_m$) x cnt times cnt = 0 to 7**

[Operands]

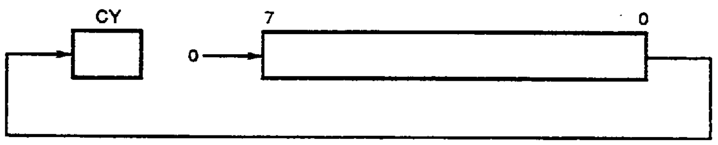
Mnemonic	Operands (dst, cnt)
SHR	r, n

[Flags]

Z	AC	CY
x	0	x

[Description]

- Shifts to the right the contents of the destination operand (dst) specified by the 1st operand cnt times as specified by the 2nd operand.
  0 is shifted into the MSB (bit 7) each time 1 bit is shifted.
- If the result of the shift operation is 0 the Z flag is set (1), otherwise the Z flag is cleared (0).
- The AC flag is always 0 regardless of the result of the shift operation.
- The final data shifted out of the LSB (bit 0) as a result of the shift operation is set in the CY flag.
- * If the second operand (cnt) is 0, no processing is performed. (Z, AC, and CY flags also do not change.)
- This instruction gives the same result as dividing the destination operand (dst) by 2^{cnt}.



[Coding example]

SHR H, 2 ; Shifts the contents of the H register 2 bits to the right.

SHL — Shift Left (Logical) Byte Data Logical Left Shift

[Instruction format] **SHL dst, cnt**

[Operation] **(CY <- dst_7, dst_0 <- 0, dst_{m+1} <- dst_m) x cnt times cnt = 0 to 7**

[Operands]

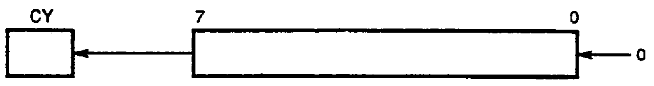
Mnemonic	Operands (dst, cnt)
SHL	r, n

[Flags]

Z	AC	CY
x	0	x

[Description]

- Shifts to the left the contents of the destination operand (dst) specified by the 1st operand cnt times as specified by the 2nd operand.
  0 is shifted into the LSB (bit 0) each time 1 bit is shifted.
- If the result of the shift operation is 0 the Z flag is set (1), otherwise the Z flag is cleared (0).
- The AC flag is always 0 regardless of the result of the shift operation.
- The final data shifted out of the MSB (bit 7) as a result of the shift operation is set in the CY flag.
- If the second operand (cnt) is 0, no processing is performed. (Z, AC, and CY flags also do not change.) ★
  To perform a one-bit left shift, execution time can be reduced by using ADD r, r.



[Coding example]

SHL E, 1 ; Shifts the contents of the E register 1 bit to the left.

SHRW

Shift Right (Logical) Word
Word Data Logical Right Shift

[Instruction format] **SHRW dst, cnt**

[Operation] **(CY <- dst_0, dst_{15} <- 0, dst_{m-1} <- dst_m) x cnt times cnt = 0 to 7**

[Operands]

Mnemonic	Operands (dst, cnt)
SHRW	rp, n

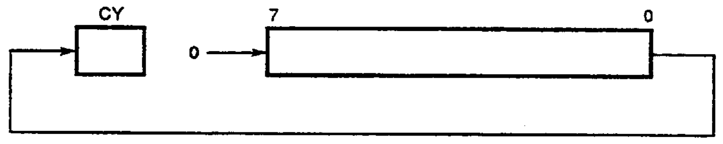
[Flags]

Z	AC	CY
x	0	x

[Description]

- Shifts to the right the contents of the destination operand (dst) specified by the 1st operand cnt times as specified by the 2nd operand.
  0 is shifted into the MSB (bit 15) each time 1 bit is shifted.
- If the result of the shift operation is 0 the Z flag is set (1), otherwise the Z flag is cleared (0).
- The AC flag is always 0 regardless of the result of the shift operation.
- The final data shifted out of the LSB (bit 0) as a result of the shift operation is set in the CY flag.

* • If the second operand (cnt) is 0, no processing is performed. (Z, AC, and CY flags also do not change.)
  This instruction gives the same result is dividing the destination operand (dst) by 2^{cnt}.



[Coding example]

SHRW AX, 3 ; Shifts the contents of the AX register 3 bits to the right (divides the AX register contents by 8).

SHLW

Shift Left (Logical) Word
Word Data Logical Left Shift

[Instruction format] **SHLW dst, cnt**

[Operation] **(CY <- dst_{15}, dst_0 <- 0, dst_{m+1} <- dst_m) x cnt times cnt = 0 to 7**

[Operands]

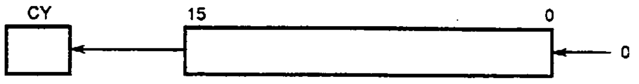
Mnemonic	Operands (dst, cnt)
SHLW	rp, n

[Flags]

Z	AC	CY
x	0	x

[Description]

- Shifts to the left the contents of the destination operand (dst) specified by the 1st operand cnt times as specified by the 2nd operand.
  0 is shifted into the LSB (bit 0) each time 1 bit is shifted.
- If the result of the shift operation is 0 the Z flag is set (1), otherwise the Z flag is cleared (0).
- The AC flag is always 0 regardless of the result of the shift operation.
- The final data shifted out of the MSB (bit 15) as a result of the shift operation is set in the CY flag.
- If the second operand (cnt) is 0, no processing is performed. (Z, AC, and CY flags also do not change.) ★



[Coding example]

SHLW E, 1 ; Shifts the contents of the E register 1 bit to the left.

ROR4

Rotate Right Digit
Digit Right Rotation

[Instruction format] **ROR4 dst**

[Operation] A_{3-0} <- $(dst)_{3-0}$, $(dst)_{7-4}$ <- A_{3-0}, $(dst)_{3-0}$ <- $(dst)_{7-4}$

[Operands]

Mnemonic	Operands (dst)
ROR4[Note]	mem1
	&mem1

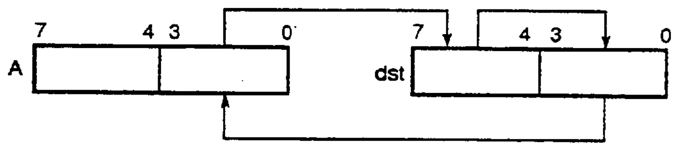
Note Cannot be used on µPD78244 sub-series EEPROM area.

[Flags]

Z	AC	CY

[Description]

- Rotates the low-order 4 bits of the A register and the destination operand (dst) 2-digit digit data (4-bit data) to the right.
  The high-order 4 bits of the A register are not changed.



[Coding example]

ROR4[HL] ; Performs digit rotation to the right of the memory contents specified by the A and HL registers.

	A		(HL)	
	7 4	3 0	7 4	3 0
Before execution	1010	0011	1100	0101
After execution	1010	0101	0011	1100

ROL4 — Rotate Left Digit / Digit Left Rotation

[Instruction format] **ROL4 dst**

[Operation] $A_{3-0} \leftarrow (dst)_{7-4}, (dst)_{3-0} \leftarrow A_{3-0}, (dst)_{7-4} \leftarrow (dst)_{3-0}$

[Operands]

Mnemonic	Operands (dst)
ROL4 Note	mem1
	&mem1

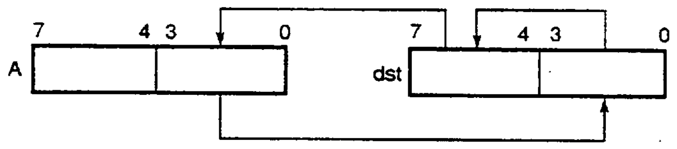
Note Cannot be used on µPD78244 sub-series EEPROM area.

[Flags]

Z	AC	CY

[Description]

- Rotates the low-order 4 bits of the A register and the destination operand (dst) 2-digit data (4-bit data) to the left.
  The high-order 4 bits of the A register are not changed.



[Coding example]

ROL4[DE+54H] ; Performs digit rotation to the left of the A register and the memory contents of address (DE register contents + 54H).

	A		(DE + 54H)	
	7 4	3 0	7 4	3 0
Before execution	0001	0010	0100	1000
After execution	0001	0100	1000	0010

6427525 0081838 706

8.8 BCD ADJUSTMENT INSTRUCTIONS

BCD adjustment instructions are as follows:

ADJBA

Decimal Adjust Register for Addition
Decimal Adjustment of Addition Result

[Instruction format] **ADJBA**

[Operation] **Decimal Adjust Accumulator for Addition**

[Operands]

None

[Flags]

Z	AC	CY
x	x	x

[Description]

- Performs A register, CY flag and AC flag decimal adjustment from the contents of the A register, CY flag and AC flag. The operation of this instruction is meaningful only when the result is stored in the A register after addition of BCD (binary-coded decimal) format data. (In other case, a meaningless operation is performed.) The adjustment method is shown in the table below.
- If the contents of the A register are 0 as a result of the adjustment the Z flag is set (1), otherwise the Z flag is cleared (0).

Condition		Operation
$A_{3-0} \leq 9$, AC = 0	$A_{7-4} \leq 9$ and CY = 0	A <- A, CY <- 0, AC <- 0
	$A_{7-4} \geq 10$ or CY = 1	A <- A + 01100000B, CY <- 1, AC <- 0
$A_{3-0} \geq 10$, AC = 0	$A_{7-4} < 9$ and CY = 0	A <- A + 00000110B, CY <- 0, AC <- 1
	$A_{7-4} \geq 9$ or CY = 1	A <- A + 01100110B, CY <- 1, AC <- 1
AC = 1	$A_{7-4} \leq 9$ and CY = 0	A <- A + 00000110B, CY <- 0, AC <- 1
	$A_{7-4} \geq 10$ or CY = 1	A <- A + 01100110B, CY <- 1, AC <- 1

ADJBS — Decimal Adjust Register for Subtraction / Decimal Adjustment of Subtraction Result

[Instruction format] **ADJBS**

[Operation] **Decimal Adjust Accumulator for Subtraction**

[Operands]

None

[Flags]

Z	AC	CY
x	x	x

[Description]

- Performs A register, CY flag and AC flag decimal adjustment from the contents of the A register, CY flag and AC flag. The operation of this instruction is meaningful only when the result is stored in the A register after subtraction of BCD (binary-coded decimal) format data. (In other cases, a meaningless operation is performed.) The adjustment method is shown in the table below.
- If the contents of the A register are 0 as a result of the adjustment the Z flag is set (1), otherwise the Z flag is cleared (0).

Condition		Operation
AC = 0	CY = 0	A <- A, CY <- 0, AC <- 0
	CY = 1	A <- A – 01100000B, CY <- 1, AC <- 0
AC = 1	CY = 0	A <- A – 00000110B, CY <- 0, AC <- 1
	CY = 1	A <- A – 01100110B, CY <- 1, AC <- 1

8.9 BIT MANIPULATION INSTRUCTIONS

Bit manipulation instructions are as follows:

6427525 0081842 137

MOV1 — Move Single Bit / 1-Bit Data Transfer

[Instruction format] **MOV1 dst, src**

[Operation] **dst <- src**

[Operands]

Mnemonic	Operands (dst, src)
MOV1	CY, saddr.bit
	CY, sfr.bit
	CY, A.bit
	CY, X.bit
	CY, PSW.bit
	saddr.bit, CY
	sfr.bit, CY
	A.bit, CY
	X.bit, CY
	PSW.bit, CY

[Flags]

dst = CY

Z	AC	CY
		x

PSW.bit

Z	AC	CY
x	x	x

Other than cases at left

Z	AC	CY

[Description]

- Transfers the bit data of the source operand (src) specified by the 2nd operand to the destination operand (dst) specified by the 1st operand.
- If the destination operand (dst) is CY or PSW.bit, only the relevant flag is changed.

[Coding example]

MOV1 P3.4, CY ; Transfers the contents of the CY flag to bit 4 of port 3.

AND1

And Single Bit
1-Bit Data Logical Product

[Instruction format] **AND1 dst, src** **AND1 dst, /src**

[Operation] **dst <- dst $\wedge$ src** **dst <- dst $\wedge$ $\overline{\text{src}}$**

[Operands]

Mnemonic	Operands (dst, src)
AND1	CY, saddr.bit
	CY, sfr.bit
	CY, A.bit
	CY, X.bit
	CY, PSW.bit

Mnemonic	Operands (dst, src)
AND1	CY, /saddr.bit
	CY, /sfr.bit
	CY, /A.bit
	CY, /X.bit
	CY, /PSW.bit

[Flags]

Z	AC	CY
		x

[Description]

- Obtains the logical product of the bit data of the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand, and stores the result in the destination operand (dst).
- If the 2nd operand is prefixed with "/", the logical product after logical negation of the source operand (src) is obtained.
- The result of the operation is stores in the CY flag (since it is the destination operand (dst)).

[Coding example]

AND1, CY, 0FE7FH.3;	Obtains the logical product of bit 3 of 0FE7FH and the CY flag, and stores the result in the CY flag.
AND1 CY, /PSW.6 ;	Obtains the logical product of the logical negation of PSW bit 6 (the Z flag) and the CY flag, and stores the result in the CY flag.

OR1

Or Single Bit
1-Bit Data Logical Sum

[Instruction format] **OR1 dst, src** **OR1 dst, /src**

[Operation] **dst <- dst $\vee$ src** **dst <- dst $\vee$ $\overline{\text{src}}$**

[Operands]

Mnemonic	Operands (dst, src)
OR1	CY, saddr.bit
	CY, sfr.bit
	CY, A.bit
	CY, X.bit
	CY, PSW.bit

Mnemonic	Operands (dst, src)
OR1	CY, /saddr.bit
	CY, /sfr.bit
	CY, /A.bit
	CY, /X.bit
	CY, /PSW.bit

[Flags]

Z	AC	CY
		x

[Description]

- Obtains the logical sum of the bit data of the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand, and stores the result in the destination operand (dst).
- If the 2nd operand is prefixed with "/", the logical sum after logical negation of the source operand (src) is obtained.
- The result of the operation is stored in the CY flag (since it is the destination operand (dst)).

[Coding example]

OR1, CY, P2.5	; Obtains the logical sum of bit 5 of port 2 and the CY flag, and stores the result in the CY flag.
OR1 CY, /X.0	; Obtains the logical sum of the logical negation of bit 0 of the X register and the CY flag, and stores the result in the CY flag.

XOR1

Exclusive Or Single Bit
1-Bit Data Exclusive Logical Sum

[Instruction format] **XOR1 dst, src**

[Operation] **dst <- dst $\veebar$ src**

[Operands]

Mnemonic	Operands (dst, src)
XOR1	CY, saddr.bit
	CY, sfr.bit
	CY, A.bit
	CY, X.bit
	CY, PSW.bit

[Flags]

Z	AC	CY
		x

[Description]

- Obtains the exclusive logical sum of the bit data of the destination operand (dst) specified by the 1st operand and the source operand (src) specified by the 2nd operand, and stores the result in the destination operand (dst).
- The result of the operation is stored in the CY flag (since it is the destination operand (dst)).

[Coding example]

XOR1, CY, A.7 ; Obtains the exclusive logical sum of bit 7 of the A register and the CY flag, and stores the result in the CY flag.

SET1 — Set Single Bit (Carry Flag) 1-Bit Data Setting

[Instruction format] **SET1 dst**

[Operation] **dst <- 1**

[Operands]

Mnemonic	Operands (dst, src)
SET1	saddr.bit
	sfr.bit
	A.bit
	X.bit
	PSW.bit
	CY

[Flags]

dst = PSW.bit

Z	AC	CY
x	x	x

dst = CY

Z	AC	CY
		1

Other than cases at left

Z	AC	CY

[Description]

- Sets (1) the destination operand (dst).
- If the destination operand (dst) is CY or PSW.bit, only the relevant flag is set (1).

[Coding example]

SET1 0FE55H.1 ; Sets (1) bit 1 of 0FE55H.

CLR1

Not Single Bit (Carry Flag)
1-Bit Data Clear

[Instruction format] **CLR1 dst**

[Operation] **dst <- 0**

[Operands]

Mnemonic	Operands (dst)
CLR1	saddr.bit
	sfr.bit
	A.bit
	X.bit
	PSW.bit
	CY

[Flags]

dst = PSW.bit

Z	AC	CY
x	x	x

dst = CY

Z	AC	CY
		0

Other than cases at left

Z	AC	CY

[Description]

- Clears (0) the destination operand (dst).
- If the destination operand (dst) is CY or PSW.bit, only the relevant flag is cleared (0).

[Coding example]

CLR1 P3.7 ; Clears (0) bit 7 of port 3.

NOT1

Not Single Bit (Carry Flag)
1-Bit Data Clear

[Instruction format] **NOT1 dst**

[Operation] **dst <- $\overline{\text{dst}}$**

[Operands]

Mnemonic	Operands (dst)
NOT1	saddr.bit
	sfr.bit
	A.bit
	X.bit
	PSW.bit
	CY

[Flags]

dst = PSW.bit

Z	AC	CY
x	x	x

dst = CY

Z	AC	CY
		x

Other than cases at left

Z	AC	CY

[Description]

- Obtains the logical negation of the bit specified by the destination operand (dst), and stores the result in the destination operand (dst).
- If the destination operand (dst) is CY or PSW.bit, only the relevant flag is changed.

[Coding example]

NOT1 A.2 ; Inverts bit 2 of the A register.

8.10 CALL/RETURN INSTRUCTIONS

Call/return instructions are as follows:

CALL — Call
Subroutine Call (16-Bit Direct/Register Indirect Specification)

[Instruction format] **CALL target**

[Operation]

$(SP-1) \leftarrow (PC+n)_H,$
$(SP-2) \leftarrow (PC+n)_L,$
$SP \leftarrow SP-2,$
$PC \leftarrow target$

n: Number of instruction bytes

[Operands]

Mnemonic	Operands (target)
CALL	!addr16
	rp

[Flags]

Z	AC	CY

[Description]

- Makes a subroutine call using a 16-bit absolute address or register indirect address.
- The start address of the next instruction (PC + n) is saved to the stack, and a branch is made to the address specified by the target operand (target).

[Coding example]

CALL !3059H ; Makes a subroutine call to 03059H.

CALLF

Call Flag
Subroutine Call (11-Bit Direct Specification)

[Instruction format] **CALLF target**

[Operation]

$(SP - 1) \leftarrow (PC + 2)_H$,
$(SP - 2) \leftarrow (PC + 2)_L$,
$SP \leftarrow SP - 2$,
$PC \leftarrow target$

[Operands]

Mnemonic	Operands (target)
CALLF	!addr11

[Flags]

Z	AC	CY

[Description]

- This subroutine call can branch only to an address in the range 00800H to 00FFFH.
- The start address of the next instruction (PC + 2) is saved to the stack, and a branch is made to an address in the range 00800H to 00FFFH.
- Only the low-order 11 address bits are specified (the high-order 5 bits are fixed at 00001B).
- Locating a subroutine in the area from 00800H to 00FFFH and using this instruction enables the program size to be reduced. The execution time is also reduced when there is a program in external memory.

[Coding example]

CALLF !0C2AH ; 00C2AH subroutine call

CALLT

Call Table
Subroutine Call (Call Table Reference)

[Instruction format] **CALLT [addr5]**

[Operation]

$(SP-1) \leftarrow (PC+1)_H,$
$(SP-2) \leftarrow (PC+1)_L,$
$SP \leftarrow SP-2,$
$PC_H \leftarrow (00000000001, addr5+1)$
$PC_L \leftarrow (00000000001, addr5)$

[Operands]

Mnemonic	Operands [addr5]
CALLT	[addr5]

[Flags]

Z	AC	CY

[Description]

- This is a call table reference subroutine call.
- The start address of the next instruction (PC + 1) is saved to the stack, and a branch is made to the address indicated by the word data in the call table (the high-order 10 bits of the address are fixed at 0000000001B, the next 5 bits are specified by addr5, and the LSB is fixed at 0).

[Coding example]

CALLT [60H] ; Subroutine call to the address indicated by the word data in addresses 00060H and 00061H.

BRK — Break Software Vectored Interrupt

[Instruction format] BRK

[Operation]

```
(SP – 1) <- PSW,
(SP – 2) <- (PC + 1)H,
(SP – 3) <- (PC + 1)L,
IE       <- 0,
SP       <- SP – 4,
PCH      <- (3FH),
PCL      <- (3EH)
```

[Operands]

None

[Flags]

Z	AC	CY

[Description]

- The software interrupt instruction.
- The PSW and the address of the next instruction (PC + 1) are saved to the stack, then the IE flag is cleared (0) and a branch is made to the address indicated by the vector address (0003EH) word data. As the IE flag is cleared (0), subsequent maskable vectored interrupts are disabled.
- Return from a software vectored interrupt generated by this instruction is performed by the RETB instruction.

RET

Return
Return from Subroutine

[Instruction format] RET

[Operation]

```
PCL  <- (SP),
PCH  <- (SP + 1),
SP   <- SP + 2
```

[Operands]

None

[Flags]

Z	AC	CY

[Description]

- This instruction is used to return from a subroutine called by the CALL, CALLF or CALLT instruction.
- The word data saved to the stack is restored to the PC, and a return is made from the subroutine.

RETI

Return from Interrupt
Return from Hardware Vectored Interrupt

[Instruction format] RETI

[Operation]

$PC_L \leftarrow (SP)$,
$PC_H \leftarrow (SP + 1)$,
$PSW \leftarrow (SP + 2)$,
$SP \leftarrow SP + 3$

[Operands]

None

[Flags]

Z	AC	CY
R	R	R

[Description]

- This instruction is used to return from a vectored interrupt.
- The data saved to the stack is restored to the PC and PSW, the NMIS flag in the IST register is cleared (0), and a return is made from the interrupt service routine.
- This instruction cannot be used to return from a software interrupt generated by the BRK instruction.
- No interrupts or macro services are acknowledged between this instruction and the next instruction to be executed.

RETB

Return from Break
Return from Software Vectored Interrupt

[Instruction format] RETB

[Operation]

```
PCL  <- (SP),
PCH  <- (SP + 1),
PSW  <- (SP + 2),
SP   <- SP + 3
```

[Operands]

None

[Flags]

Z	AC	CY
R	R	R

[Description]

- This instruction is used to return from a software interrupt generated by the BRK instruction..
- The PC and PSW saved to the stack are restored, and a return is made from the interrupt service routine.
- No interrupts or macro services are acknowledged between this instruction and the next instruction to be executed.

8.11 STACK MANIPULATION INSTRUCTIONS

Stack manipulation instructions are as follows:

PUSH

Push
Push

[Instruction format] **PUSH src**

[Operation]

When src = rp	When src = PSW or sfr
(SP – 1) <- srcH,	**(SP – 1) <- src**
(SP – 2) <- srcL,	**SP <- SP – 1**
SP <- SP – 2	

[Operands]

Mnemonic	Operands (src)
PUSH	PSW
	sfr
	rp

[Flags]

Z	AC	CY

[Description]

- Saves the data in the register specified by the source operand (src) to the stack.

[Coding example]

PUSH AX ; Saves the contents of the AX register to the stack.

POP

Pop
Pop

[Instruction format] **POP dst**

[Operation]

When dst = rp	When dst = PSW or sfr
dst_L <- (SP),	**dst <- (SP)**
dst_H <- (SP + 1),	**SP <- SP + 1**
SP <- SP + 2	

[Operands]

Mnemonic	Operands (src)
POP	PSW
	sfr
	rp

[Flags]

src = PSW

Z	AC	CY
R	R	R

Other than cases at left

Z	AC	CY

[Description]

- Restores data from the stack to the register specified by the destination operand (dst).
- If the operand is the PSW, each flag is replaced with stack data.
- No interrupts or macro services are acknowledged between the POP PSW instruction and the next instruction.

[Coding example]

POP IMK0L ; Restores stack data to the IMK0L register.

MOVW SP, src / MOVW AX, SP — Move Word / Stack Pointer/Word Data Transfer

[Instruction format] **MOVW dst, src**

[Operation] **dst <- src**

[Operands]

Mnemonic	Operands (dst, src)
MOVW	SP, #word
	SP, AX
	AX, SP

[Flags]

Z	AC	CY

[Description]

- These instructions are used to manipulate the contents of the stack pointer.
- Stores the source operand (src) specified by the 2nd operand in the destination operand (dst) specified by the 1st operand.

[Coding example]

MOVW PS, #0FE1FH ; Stores 0FE1FH in the stack pointer.

INCW SP

Increment Word
Stack Pointer Increment

[Instruction format] **INCW SP**

[Operation] **SP <- SP + 1**

[Operands]

None

[Flags]

Z	AC	CY

[Description]

- Increments the contents of the SP (stack pointer) by 1.

DECW SP

Decrement Word
Stack Pointer Decrement

[Instruction format] **DECW SP**

[Operation] **SP <- SP – 1**

[Operands]

None

[Flags]

Z	AC	CY

[Description]

- Decrements the contents of the SP (stack pointer) by 1.

8.12 UNCONDITIONAL BRANCH INSTRUCTIONS

The unconditional branch instruction is as follows:

BR ... 206

BR

Branch
Unconditional Branch

[Instruction format] **BR target**

[Operation] **PC <- target**

[Operands]

Mnemonic	Operands (target)
BR	!addr16
	rp
	$addr16

[Flags]

Z	AC	CY

[Description]
- Performs an unconditional branch.
- Transfers the target address operand (target) word data to the PC, then branches.

[Coding example]
BR AX ; Branches using the contents of the AX register as the branch address.

8.13 CONDITIONAL BRANCH INSTRUCTIONS

Conditional branch instructions are as follows:

BC / BL — Branch if Carry/Less than — Conditional Branch depending on Carry Flag (CY = 1)

[Instruction format] **BC $addr16**
BL $addr16

[Operation] **PC <- PC + 2 + jdisp8 if CY = 1**

[Operands]

Mnemonic	Operands ($addr16)
BC	$addr16
BL	

[Flags]

Z	AC	CY

[Description]

- When CY = 1, branches to the address specified by the operand.
  When CY = 0, no processing is performed and the next instruction is executed.
- The operation of the BC instruction and the BL instruction is the same. Differences in their use are as follows:
  - BC instruction : Checks whether a carry has been generated after an addition instruction.
    Determines the result of bit manipulation.
  - BL instruction : Checks whether a borrow has been generated after a subtraction instruction.
    After a compare instruction, checks whether or not the 1st operand of the compare instruction is smaller.

[Coding example]

BC $300H ; Branches to 00300H if CY = 1 (the start of this instruction is in the address range from 0027FH to 0037EH).

BNC / BNL — Branch if Not Carry/Less than — Conditional Branch depending on Carry Flag (CY = 0)

[Instruction format] **BNC $addr16**
BNL $addr16

[Operation] **PC <- PC + 2 + jdisp8 if CY = 0**

[Operands]

Mnemonic	Operands ($addr16)
BNC	$addr16
BNL	

[Flags]

Z	AC	CY

[Description]

- When CY = 0, branches to the address specified by the operand.
  When CY = 1, no processing is performed and the next instruction is executed.
- The operation of the BNC instruction and the BNL instruction is the same. Differences in their use are as follows:
  - BNC instruction : Checks whether a carry has been generated after an addition instruction. Determines the result of bit manipulation.
  - BNL instruction : Checks whether a borrow has been generated after a subtraction instruction. After a compare instruction, checks whether or not the 1st operand of the compare instruction is smaller.

[Coding examples]

CMP A, B	; Compares the size of the A register contents and B register contents.
BNL #1500H	; Branches to 01500H if the contents of the A register are not smaller than the contents of the B register (the start of this instruction is in the address range from 0147FH to 0157EH).

BZ / BE — Branch if Zero/Equal — Conditional Branch depending on Zero Flag (Z = 1)

[Instruction format] **BZ $addr16**
BE $addr16

[Operation] **PC <- PC + 2 + jdisp8 if Z = 1**

[Operands]

Mnemonic	Operands ($addr16)
BZ	$addr16
BE	

[Flags]

Z	AC	CY

[Description]

- When Z = 1, branches to the address specified by the operand.
  When Z = 0, no processing is performed and the next instruction is executed.
- The operation of the BZ instruction and the BE instruction is the same. Differences in their use are as follows:
  - BZ instruction : Checks whether the result of an addition, subtraction or increment/decrement instruction or an 8-bit logical operation is 0.
  - BE instruction : Checks for a match after a compare instruction.

[Coding example]

DEC B
BZ $3C5H ; Branches to 003C5H if the B register contents are 0 (the start of this instruction is in the address range from 00344H to 00443H).

BNZ / BNE — Branch if Not Zero/Not Equal

Conditional Branch depending on Zero Flag (Z = 0)

[Instruction format] **BNZ $addr16**
BNE $addr16

[Operation] **PC <- PC + 2 + jdisp8 if Z = 0**

[Operands]

Mnemonic	Operands ($addr16)
BNZ	$addr16
BNE	

[Flags]

Z	AC	CY

[Description]

- When Z = 0, branches to the address specified by the operand.
  When Z = 1, no processing is performed and the next instruction is executed.
- The operation of the BNZ instruction is the same. Differences in their use are as follows:
  - BNZ instruction: Checks whether the result of an addition, subtraction or increment/decrement instruction or an 8-bit logical operation is 0.
  - BNE instruction: Checks for a match after a compare instruction.

[Coding example]

CMP A, #55H
BNE $0A39H ; Branches to 00A39H if the A register contents are not 0055H (the start of this instruction is in the address range from 009B8H to 00AB7H).

BT — Branch if True — Conditional Branch depending on Bit Test (Byte Data Bit = 1)

[Instruction format] **BT bit, $addr16**

[Operation] **PC <- PC + b + jdisp8 if bit = 1**

[Operands]

Mnemonic	Operands (bit, $addr16)	b (Number of bytes)
BT	saddr.bit, $addr16	3
	sfr.bit, $addr16	4
	A.bit, $addr16	3
	X.bit, $addr16	3
	PSW.bit, $addr16	3

[Flags]

Z	AC	CY

[Description]

- If the contents of the 1st operand (bit) are set (1), branches to the address specified by the 2nd operand ($addr16).
  If the contents of the 1st operand (bit) are not set (1), no processing is performed and the next instruction is executed.

[Coding example]

BT 0FE47H.3, $55CH ; Branches to 0055CH if bit 3 of address 0FE47H is 1 (the start of this instruction is in the address range from 004DAH to 005D9H).

BF — Branch if False
Conditional Branch depending on Bit Test (Byte Data Bit = 0)

[Instruction format] **BF bit, $addr16**

[Operation] **PC <- PC + b + jdisp8 if bit = 0**

[Operands]

Mnemonic	Operands (bit, $addr16)	b (Number of bytes)
BF	saddr.bit, $addr16	4
	sfr.bit, $addr16	4
	A.bit, $addr16	3
	X.bit, $addr16	3
	PSW.bit, $addr16	3

[Flags]

Z	AC	CY

[Description]

- If the contents of the 1st operand (bit) are cleared (0), branches to the address specified by the 2nd operand ($addr16).

  If the contents of the 1st operand (bit) are not cleared (0), no processing is performed and the next instruction is executed.

[Coding example]

BF P2.2, $1549H ; Branches to 01549H if bit 2 of port 2 is 0 (the start of this instruction is in the address range from 014C6H to 015C5H).

BTCLR

Branch if True and Clear
Conditional Branch and Clear depending on Bit Test
(Byte Data Bit = 1)

[Instruction format] **BTCLR bit, $addr16**

[Operation] **PC <- PC + b + jdisp8 if bit = 1, then bit <- 0**

[Operands]

Mnemonic	Operands (bit, $addr16)	b (Number of bytes)
BTCLR	saddr.bit, $addr16	4
	sfr.bit, $addr16	4
	A.bit, $addr16	3
	X.bit, $addr16	3
	PSW.bit, $addr16	3

[Flags]

bit = PSW.bit

Z	AC	CY
x	x	x

Other than cases at left

Z	AC	CY

[Description]

- If the contents of the 1st operand (bit) are set (1), clears (0) the contents of the 1st operand (bit) and branches to the address specified by the 2nd operand.
  If the contents of the 1st operand (bit) are not set (1), no processing is performed and the next instruction is executed.
- If the 1st operand (bit) is PSW.bit, only the contents of the relevant flag are cleared (0).

[Coding example]

BTCLR PSW.0, $356H ; If PSW bit 0 (CY flag) is 1, clears the CY flag and branches to address 00356H (the start of this instruction is in the address range from 002D4H to 003D3H).

DBNZ

Decrement and Branch if Not Zero
Conditional Loop (R1 ≠ 0)

[Instruction format] **DBNZ dst, $addr16**

[Operation] **dst <- dst − 1,**
then PC <- PC + b + jdisp16 if dst R1 ≠ 0

[Operands]

Mnemonic	Operands (bit, $addr16)	b (Number of bytes)
DBNZ	r1, ← $addr16	2
	saddr, $addr16	3

[Flags]

Z	AC	CY

[Description]

- Decrements by 1 the contents of the destination operand (dst) specified by the 1st operand, and branches to the destination operand (dst).
- If the result of decrementing the destination operand (dst) by 1 is not 0, the instruction branches to the address indicated by the 2nd operand ($addr16).
  If the result of decrementing the destination operand (dst) by 1 is 0, no processing is performed and the next instruction is executed.
- Flags are not changed.

[Coding example]

DBNZ B, $1215H ; Decrements the contents of the B register, and if the result is not 0, branches to 001215H (the start of this instruction is in the address range from 001194H to 001293H).

6427525 0081874 747

8.14 CPU CONTROL INSTRUCTIONS

CPU control instructions are as follows:

MOV STBC, #byte

Move
Standby Mode Setting

[Instruction format] **MOV STBC #byte**

[Operation] **STBC <- byte**

[Operands]

Mnemonic	Operands
MOV	STBC, #byte

[Flags]

Z	AC	CY

[Description]

- This is a dedicated write instruction for the standby control register (STBC). This instruction writes the immediate data specified by the 2nd operand to STBC.
  Only this instruction can be used to write to the STBC register.
- This instruction has a special format, and in addition to the immediate data used in performing the write, the operation code must also provide data resulting from the logical negation of that value (see figure below).
  (This is generated automatically by NEC assemblers.)

- Instruction code format

```
0 0 0 0   1 0 0 1
1 1 0 0   0 0 0 0
```

$\leftarrow \overline{\text{byte}} \rightarrow$

$\leftarrow \text{byte} \rightarrow$

- The CPU checks the data obtained by logical negation of the immediate data to be written, and if correct, performs the write. If the data is incorrect, the write is not performed and the next instruction is executed.

[Coding example]

MOV STBC, #2 ; Writes 2 to STBC (sets STOP mode).

SEL RBn

Select Register Bank
Register Bank Selection

[Instruction format] **SEL RBn**

[Operation] **RBS0, RBS1 <- n; (n = 0 to 3)**

[Operands]

Mnemonic	Operands (RBn)
SEL	RBn

[Flags]

Z	AC	CY

[Description]

- Selects the register bank specified by the operand (RBn) as the register bank to be used by the next and subsequent instructions.
- RBn comprises RB0 to RB3.

[Coding example]

SEL RB2 ; Selects register bank 2 as the register bank to be used from the next instruction onward.

6427525 0081877 456

NOP

No Operation
No Operation

[Instruction format] **NOP**

[Operation] **no operation**

[Operands]

None

[Flags]

Z	AC	CY

[Description]

- Expends time without performing any processing.

EI

Enable interrupt
Interrupt Enabling

[Instruction format] **EI**

[Operation] **IE <- 1**

[Operands]

None

[Flags]

Z	AC	CY

[Description]

- Sets the acknowledgment enabled state for maskable interrupts (sets (1) the interrupt enable flag (IE)).
- No interrupts or macro service requests are acknowledged between this instruction and the next instruction.
- Acknowledgment of vectored interrupts from other sources can be disabled even when this instruction is executed. Refer to individual documentation for details.

DI

Disable interrupt
Interrupt Disabling

[Instruction format] DI

[Operation] IE <- 0

[Operands]

None

[Flags]

Z	AC	CY

[Description]

- Disables acknowledgment of vectored maskable interrupts (clears (0) the interrupt enable flag (IE)).
- No interrupts or macro service requests are acknowledged between this instruction and the next instruction.
- Refer to individual documentation for details of interrupt servicing.