900, 900/L, 900/H, 900/L1, 900/H2 CPU Core Different Points

There are 5 type CPU core: 900, 900/L, 900/H, 900/L1, and 900/H2 in TLCS-900 series and they are different from following points.

Table 1 Differences between CPUs

CPU Content of Difference	900	900/L	900/H, 900/L1	900H2
Maximum address bus width	24 bits	←	←	←
Maximum data bus width	16 bits	←	←	32 bits
Instruction queue buffer	4 bytes	←	←	12 bytes
Instruction set	TLCS-900	Following instructions deleted from TLCS-900 NORMAL MAX Following instructions added MIN	Following instructions deleted from TLCS-900 NORMAL MAX	Following instructions deleted from TLCS-900 NORMAL MAX LDX
Code fetch during branch instruction execution	Jump addres code is fetched only when branch condition is true	←	←	Jump address code is always prefetched irrespective of branch condition
Micro DMA	4 channels	←	←	8 channels
CPU privilege mode	System mode and normal mode	System mode only	←	←
CPU register mode	MIN mode and MAX mode (MIN mode after reset)	← (MAX mode after reset)	MAX mode only	←
Interrupt method	Restart method	Vector method	←	←
Normal stack pointer, XNSP	Available	N/A	←	←
Interrupt nesting counter, INTNEST (used mainly for the OS)	N/A	Available	←	←

1. Outline

The TLCS-900/L1 series has an original Toshiba high-performance 16-bit CPU. Combining the CPU with various I/O function blocks (such as timers, serial I/Os, ADs) creates broad possibilities in application fields.

The TLCS-900/L1 CPU, being 16-bit CPU, has a 32-bit/16-bit register bank configuration, therefore it is suitable as an embedded controller.

The TLCS-900/L1 CPU features are as follows :

(1) TLCS-90 extended architecture

- Upward compatibility on mnemonic and register set levels

(2) General-purpose registers

- All 8 registers usable as accumulator

(3) Register bank system

- four 32-bit register banks

(4) 16M-byte linear address space ; 9 types addressing modes

(5) Dynamic bus sizing system

- Can consist 8-/16-bit external data bus together

(6) Orthogonal instruction sets

- 8-/16-/32-bit data transfer/arithmetic instructions
- 16-bit multiplication/division
  16×16 to 32-bits (signed/unsigned)
  $32 \div 16$ to 16-bits: remainder 16-bits (signed/unsigned)
- Bit processing including bit arithmetic
- Supporting instruction for C compiler
- Filter calculations: multiplication-addition arithmetic, modulo increment instruction

(7) High-speed processing

- Pipeline system with 4-byte instruction queue buffer
- 32-bit ALU

2. CPU Operating Modes

The 900/L1 has only system mode.

In system mode, there are no restrictions on using instructions or registers.

The CPU resources effective in system mode are as follows:

(1) General-purpose registers

- Four 32-bit general-purpose registers × 4 banks
- Four 32-bit general-purpose registers (including system stack pointer: XSP)

(2) Status register (SR)

(3) Program counter (PC): 32 bits

(4) Control register: parameter register for micro DMA, etc.

(5) All CPU instructions

(6) All built-in I/O registers

(7) All built-in memories

3. Registers

3.1 Register Structure · · · 16-Mbyte program area/16-Mbyte data area

Figure 3.1.1 illustrates the format of registers.

Four 32-bit general-purpose registers × 4 banks

+

Four 32-bit general-purpose registers

+

32-bit program counter

+

Status register

Register mode changing

The <MAX> bit in status register (SR) is initialized to 1 and set to Maximum mode by resetting. The 900/L1 has only Maximum mode.

Stack Pointer

The stack pointer (SP) is provided for only System mode (XSP). The System stack pointer (XSP) is set to 100H by resetting.

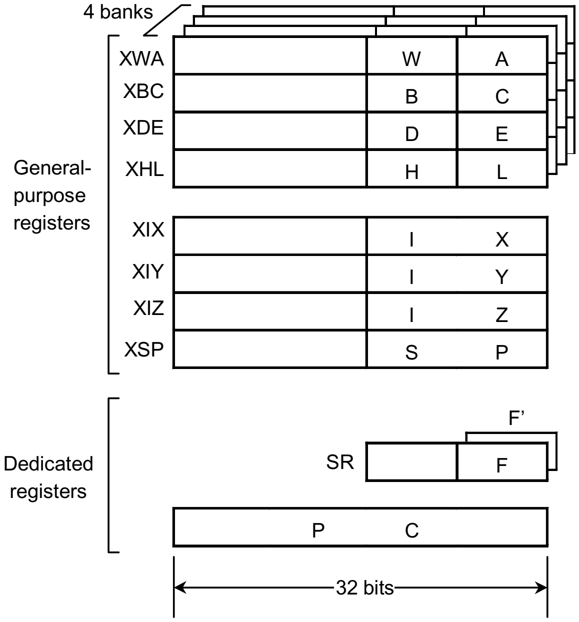


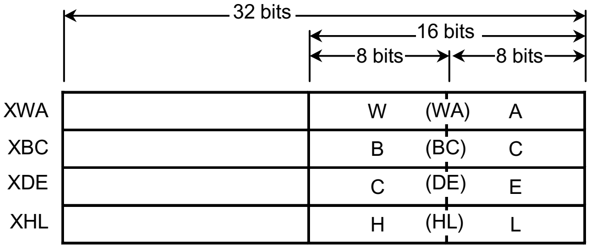
Figure 3.1.1 Register format (16-Mbyte program area)

3.2 Register Details

3.2.1 General-purpose bank registers

In maximum mode, the following four 32-bit general-purpose registers consisting of 4 banks can be used. The register format in a bank is shown below.

Four 32-bit registers (XWA, XBC, XDE, and XHL) are general-purpose registers and can be used as an accumulators and index registers. They can also be used as 16-bit registers (WA, BC, DE, and HL), in which case, the lower 16 bits of the 32-bit registers are assigned.



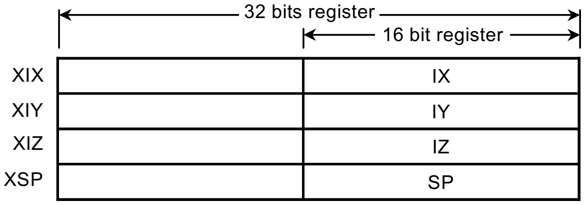
Note: Round brackets ( ) signify 16-bit registers.

16-bit registers can be used as accumulators, index registers in index addressing mode, and displacement registers. They can also be used as two 8-bit general-purpose registers (W, A, B, C, D, E, H, and L) to function for example as accumulators.

3.2.2 32-bit general-purpose registers

The TLCS-900 series has four 32-bit general-purpose registers (XIX, XIY, XIZ, and XSP). The register format is shown below.

These registers can also be used as accumulators, index registers, and displacement registers. They can be used either as 16-bit, or 8-bit registers. Names when registers are used as 8-bit registers are listed later.



Stack Pointer

The XSP register is utilized for stack pointer. It is used when the interrupt is occured or CALL, RET instruction are executed. The stack pointer (XSP) is set to 100H by resetting.

3.2.3 Status Register (SR)

The status register contains flags indicating the status (operating mode, register format, etc.) of the CPU and operation results. This register consists of two parts. The upper byte of the status register (bits 8 to 15) indicates the CPU status. The lower byte (bits 0 to 7) are referred to as the flag register (F). This indicates the status of the operation result. The TLCS-900 series has two flag registers (F and F'). They can be switched using the EX instruction.

(1) Upper Byte of Status Register

15	14	13	12	11	10	9	8
SYSM	IFF2	IFF1	IFF0	MAX	RFP2	RFP1	RFP0

1. SYSM (System Mode)

   Indicates the CPU operating mode, system or normal. 900/L1 has only system mode.

0	Normal mode
1	System mode (900/L1 has only this mode.)

2. IFF2 to IFF0 (Interrupt mask Flip-Flop2 to 0)

   Mask registers with interrupt levels from 1 to 7. Level 7 has the highest priority.

   Initialized to 111 by reset.

000	Enables interrupts with level 1 or higher.	Same
001	Enables interrupts with level 1 or higher.	
010	Enables interrupts with level 2 or higher.	
011	Enables interrupts with level 3 or higher.	
100	Enables interrupts with level 4 or higher.	
101	Enables interrupts with level 5 or higher.	
110	Enables interrupts with level 6 or higher.	
111	Enables interrupts with level 7 only (non-maskable interrupt).	

Any value can be set using the EI instruction.

When an interrupt is received, the mask register sets a value higher by 1 than the interrupt level received. When an interrupt with level 7 is received, 111 is set. Unlike with the TLCS-90 series, the EI instruction becomes effective immediately after execution.

3. MAX ($\overline{\text{MINimum}}$/ MAXimum)

Bit used to specify the register mode which determines the sizes of the register banks and the program counter.

0	Minimum mode
1	Maximum mode (900/L1 has only this mode.)

Initialized to 1 (maximum mode) for 900/L1 by reset.

The minimum mode is not support for 900/L1. Therefore, do not write 0 to this bit.

4. RFP2 to RFP0 (Register File Pointer2 to 0)

Indicates the number of register file (register bank) currently being used. Initialized to 000 by reset.

The values in these registers can be operated on using the following three instructions. RFP2 is fixed to 0 in maximum mode. It remains 0 even if an attempt to change it to 1 using following instructions.

- LDF imm ; RFP ← imm (0 to 3)
- INCF ; RFP ← RFP + 1
- DECF ; RFP ← RFP − 1

(2) Flag Register, F

7	6	5	4	3	2	1	0	
S	Z	0	H	0	V	N	C	:R/W

1. S (Sign flag)

   1 is set when the operation result is negative, 0 when positive.

   (The value of the most significant bit of the operation result is copied.)

2. Z (Zero flag)

   1 is set when the operation result is zero, otherwise 0.

3. H (Half carry flag)

   1 is set when a carry or borrow from bit 3 to bit 4 occurs as a result of the operation, otherwise 0. With a 32-bit operation instruction, an undefined value is set.

4. V (Parity/over-flow flag)

   Indicates either parity or overflow, depending on the operation type.

   Parity (P): 0 is set when the number of bits set to 1 is odd, 1 when even.
   An undefined value is set with a 32-bit operation instruction.

   Overflow (V): 0 is set if no overflow, if overflow 1.

5. N (Negative)

   ADD/SUB flag

   0 is set after an addition instruction such as ADD is executed, 1 after a subtraction instruction such as SUB.

   Used when executing the DAA (decimal addition adjust accumulator) instruction.

6. C (Carry flag)

   1 is set when a carry or borrow occurs, otherwise 0.

Read and write process of status register

Read from bits 0 to 15	1. { PUSH SR POP dst
Write to bits 0 to 15	1. POP SR
Only bit 15 <SYSM>	1 is always set, because 900/L1 CPU has only system mode.
Only bits 14 to 12 <IFF2:0>	1. EInum A value of num is written.
Only bit 11 <MAX>	The minimum mode is not support for 900/L1. Therefore, do not write 0 to this bit.
Only bits 10 to 8 <RFP2:0>	1. LDF imm 2. INCF 3. DECF
Only bits 7 to 0	1. PUSH F / POP F 2. EX F, F’ 3. A flag is set indirectly by executing arithmetic instructions etc.

3.2.4 Program Counter (PC)

The program counter is a pointer indicating the memory address to be executed next.

In maximum mode, the program counter consists of 32 bits. The size of the program area depends on the number of the address pins that the product has. With 24 address pins (A0 to A23), a maximum program area of 16 Mbytes can be accessed as a linear address space. In this case, the upper 8 bits of the program counter (bits 24 to 31) are ignored.

PC after reset

The 900/L1 reads a value of a reset vector from a vector base address by reset and sets the value into a program counter. Then, program after the vector specified by the program counter are executed.

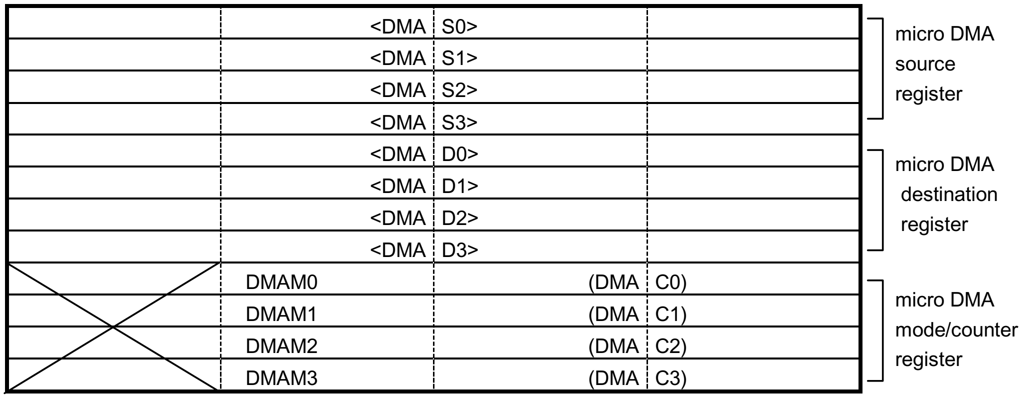
The vector base address is depending on products. They are given below.

Vector Base Address	PC setting after reset	
0FFFF00H	PC (7:0) ← 0FFFF00H PC (15:8) ← 0FFFF01H PC (23:16) ← 0FFFF02H	value of address

3.2.5 Control registers (CR)

The control registers consist of registers used to control micro DMA operation and an interrupt nesting counter. Control registers can be accessed by using the LDC instruction.

Control registers are illustrated below.



	<DMA	S0>		micro DMA source register
	<DMA	S1>		
	<DMA	S2>		
	<DMA	S3>		
	<DMA	D0>		micro DMA destination register
	<DMA	D1>		
	<DMA	D2>		
	<DMA	D3>		
	DMAM0		(DMA | C0)	micro DMA mode/counter register
	DMAM1		(DMA | C1)	
	DMAM2		(DMA | C2)	
	DMAM3		(DMA | C3)	

(INT	NEST)

Interrupt Nesting Counter

( ) : Word register name (16 bits)

< > : Long word register name (32 bits)

For micro DMA, refer to Chapter 4 TLCS-900/L1 LSI Devices.

3.3 Register Bank Switching

Register banks are classified into the following three types.

Current bank registers

Previous bank registers

Absolute bank registers

The current bank is indicated by the register file pointer, <RFP>, (status register bits 8 to 10). The registers in the current bank are used as general-purpose registers, as described in the previous section. By changing the contents of the <RFP>, another register bank becomes the current register bank.

The previous bank is indicated by the value obtained by subtracting 1 from the <RFP>. For example, if the current bank is bank 3, bank 2 is the previous bank. The names of registers in the previous bank are indicated with a dash (WA', BC', DE', HL'). The EX instruction (EX A,A') is used to switch between current and previous banks.

All bank registers, including the current and previous ones, have a numerical value (absolute bank number) to indicate the bank. With a register name which includes a numerical value such as RW0, RA0, etc., all bank registers can be used. These registers (that is, all registers) are called absolute bank registers.

The TLCS-900 series CPU is designed to perform optimally when the current bank registers are operated as the working registers. In other words, if the CPU uses other bank registers, its performance degrades somewhat. In order to obtain maximum CPU efficiency, the TLCS-900 series has a function which easily switches register banks.

The bank switching function provides the following advantages:

- Optimum CPU operating efficiency
- Reduced programming size (Object codes)
- Higher response speed and reduced programming size when used as a context switch for an interrupt service routine.

Bank switching is performed by the instructions listed below.

LDF imm	:	Sets the contents of the immediate value in <RFP>. imm: 0 to 3
INCF	:	Increments <RFP> by 1.
DECF	:	Decrements <RFP> by 1.

The immediate values used by the LDF instruction are from 0 to 3. If a carry or borrow occurs when the INCF or DECF instruction is executed, it is ignored. The value of the <RFP> rotates. For example, if the INCF instruction is executed with bank 3, the result is bank 0. If the DECF instruction is executed with bank 0, the result is bank 3. Note that careless execution of the INCF or DECF instruction may destroy the contents of the register bank.

- Example of Register Bank Usage

The TLCS-900 series registers are formatted in banks. Banks can be used for processing objectives or interrupt levels. Two examples are given below.

<Example 1> When assigning register banks to interrupt processing routines.

Register bank 0 = Used for the main program and interrupt processing other than that shown below.

Register bank 1 = Used for processing INT0 .

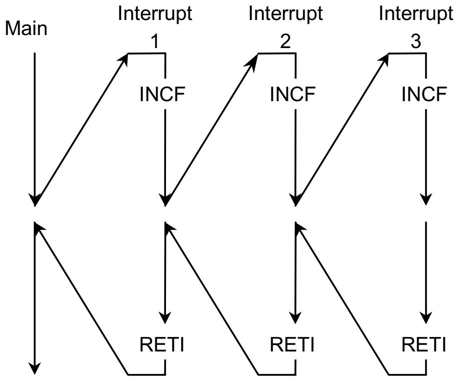
Register bank 2 = Used for processing timer 0.

Register bank 3 = Used for processing timer 1.

For example, if a timer 1 interrupt occurs during main program execution, processing jumps to a subroutine as follows. PUSH/POP processing for the register is unnecessary.

```
LDF 3     ;  Sets register bank to 3.
  :
  :
RETI      ;  Returns to previous status including <RFP>.
```

<Example 2> When assigning register banks to their appropriate interrupt level nesting.



Note 1: In the above example, when interrupt nesting exceeds the number of register banks (4), the <RFP> becomes 000 and the contents of register bank 0 are destroyed.

Note 2: The INCF instruction is used to execute <RFP> ← <RFP> + 1.

3.4 Accessing General-purpose Registers

The register access code is formatted in a varied code length on byte basis. The current bank registers can be accessed by the shortest code length. All general-purpose registers can be accessed by an instruction code which is 1 byte longer. General-purpose registers are as follows.

1. General-purpose registers in current bank

QW	(Q	WA)	QA	<X	WA >	W	(W	A)	A
QB	(Q	BC)	QC	<X	BC >	B	(B	C)	C
QD	(Q	DE)	QE	<X	DE >	D	(D	E)	E
QH	(Q	HL)	QL	<X	HL >	H	(H	L)	L

( ) : Word register name (16 bits)
< > : Long word register name (32 bits)

2. General-purpose registers in previous bank

QW’	(Q	WA’)	QA’	<X	WA’ >	W’	(W	A’)	A’
QB’	(Q	BC’)	QC’	<X	BC’ >	B’	(B	C’)	C’
QD’	(Q	DE’)	QE’	<X	DE’ >	D’	(D	E’)	E’
QH’	(Q	HL’)	QL’	<X	HL’ >	H’	(H	L’)	L’

3. 32-bit general-purpose registers

QIXH	(Q	IX)	QIXL	<X	IX >	IXH	(I	X)	IXL
QIYH	(Q	IY)	QIYL	<X	IY >	IYH	(I	Y)	IYL
QIZH	(Q	IZ)	QIZL	<X	IZ >	IZH	(I	Z)	IZL
QSPH	(Q	SP)	QSPL	<X	SP >	SPH	(S	P)	SPL

4. Absolute bank registers

QW0	(QWA	0)	QA0	<XWA	0>	RW0	(RWA	0)	RA0	Bank0
QB0	(QBC	0)	QC0	<XBC	0>	RB0	(RBC	0)	RC0	
QD0	(QDE	0)	QE0	<XDE	0>	RD0	(RDE	0)	RE0	
QH0	(QHL	0)	QL0	<XHL	0>	RH0	(RHL	0)	RL0	
QW1	(QWA	1)	QA1	<XWA	1>	RW1	(RWA	1)	RA1	Bank1
QB1	(QBC	1)	QC1	<XBC	1>	RB1	(RBC	1)	RC1	
QD1	(QDE	1)	QE1	<XDE	1>	RD1	(RDE	1)	RE1	
QH1	(QHL	1)	QL1	<XHL	1>	RH1	(RHL	1)	RL1	
QW2	(QWA	2)	QA2	<XWA	2>	RW2	(RWA	2)	RA2	Bank2
QB2	(QBC	2)	QC2	<XBC	2>	RB2	(RBC	2)	RC2	
QD2	(QDE	2)	QE2	<XDE	2>	RD2	(RDE	2)	RE2	
QH2	(QHL	2)	QL2	<XHL	2>	RH2	(RHL	2)	RL2	
QW3	(QWA	3)	QA3	<XWA	3>	RW3	(RWA	3)	RA3	Bank3
QB3	(QBC	3)	QC3	<XBC	3>	RB3	(RBC	3)	RC3	
QD3	(QDE	3)	QE3	<XDE	3>	RD3	(RDE	3)	RE3	
QH3	(QHL	3)	QL3	<XHL	3>	RH3	(RHL	3)	RL3	

( ) : Word register name (16 bits)

< > : Long word register name (32 bits)

4. Addressing Modes

The TLCS-900/L1 series has nine addressing modes. These are combined with most instructions to improve CPU processing capabilities.

TLCS-900/L1 series addressing modes are listed below. They cover the entire TLCS-90 addressing modes.

No.	Addressing mode	Description
1.	Register	reg8 reg16 reg32
2.	Immediate	n8 n16 n32
3.	Register indirect	(reg)
4.	Register indirect pre-decrement	(–reg)
5.	Register indirect post-increment	(reg+)
6.	Index	(reg + d8) (reg + d16)
7.	Register index	(reg + reg8) (reg + reg16)
8.	Absolute (Direct addressing mode)	(n8) (n16) (n24)
9.	Relative	(PC + d8) (PC + d16)

reg 8: All 8-bit registers such as W, A, B, C, D, E, H, L, etc.

reg 16: All 16-bit registers such as WA, BC, DE, HL, IX, IY, IZ, SP, etc.

reg 32: All 32-bit registers such as XWA, WBC, XDE, XHL, XIX, XIY, XIZ, XSP, etc.

reg: All 32-bit registers such as XWA, WBC, XDE, XHL, XIX, XIY, XIZ, XSP, etc.

d8: 8-bit displacement (–80H to + 7FH)

d16: 16-bit displacement (–8000H to + 7FFFH)

n8: 8-bit constant (00H to FFH)

n16: 16-bit constant (0000H to FFFFH)

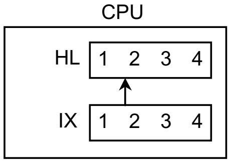
n32: 32-bit constant (00000000H to FFFFFFFFH)

Note 1: Relative addressing mode can only be used with the following instructions:
LDAR, JR, JRL, DJNZ, and CALR

(1) Register Addressing Mode

In this mode, the operand is the specified register.

Example: LD HL, IX

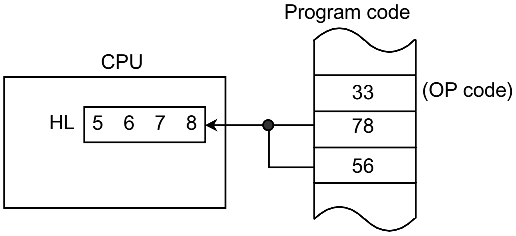


The IX register contents, 1234H, are loaded to the HL register.

(2) Immediate Addressing Mode

In this mode, the operand is in the instruction code.

Example: LD HL, 5678H

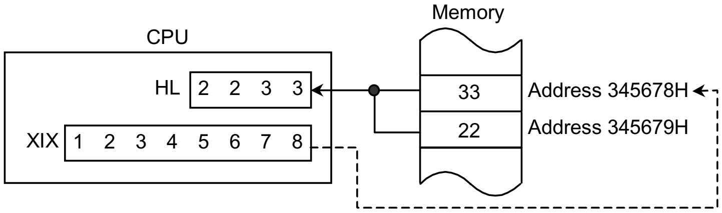


The immediate data, 5678H, is loaded to the HL register.

(3) Register Indirect Addressing Mode

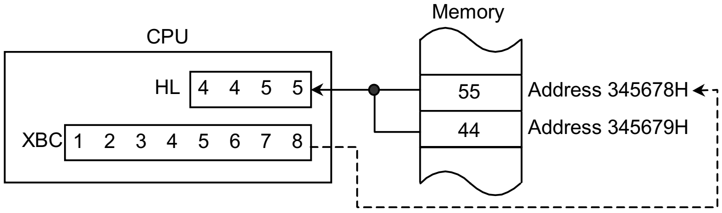
In this mode, the operand is the memory address specified by the contents of the register.

Example 1: LD, HL, (XIX)



Memory data, 2233H, at address 345678H is loaded to the HL register.

Example 2: LD HL, (XBC)

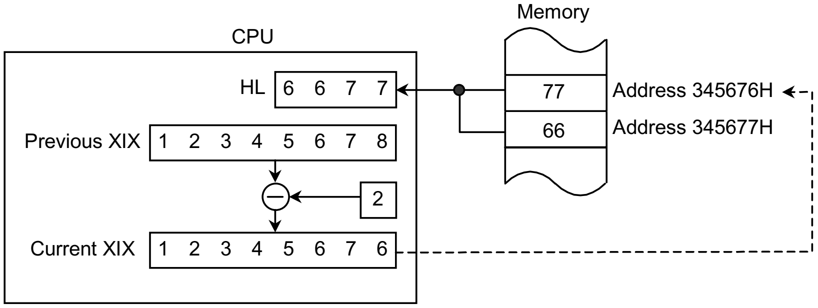


If a bank register (XWA, XBC, XDE, or XHL) is used for addressing, the values of bits 0 to 23 are output to the address bus.

(4) Register Indirect Pre-decrement Addressing Mode

In this mode, the contents of the register is decremented by the pre-decrement values. In this case, the operand is the memory address specified by the decremented register.

Example 1: LD HL, (–XIX)



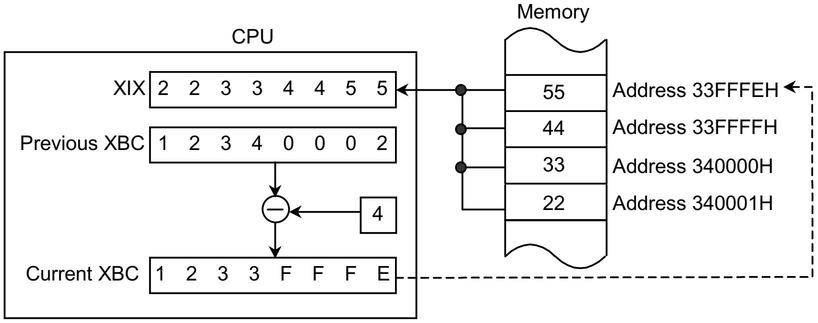
The pre-decrement values are as follows:

When the size of the operand is one byte (8 bits): –1

When the size of the operand is one word (16 bits): –2

When the size of the operand is one long word (32 bits): –4

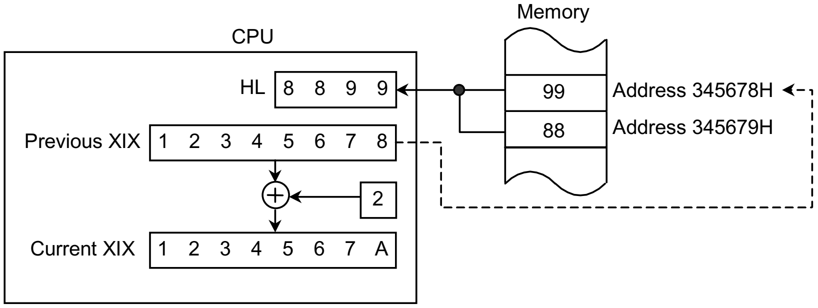
Example 2: LD XIX, (–XBC)



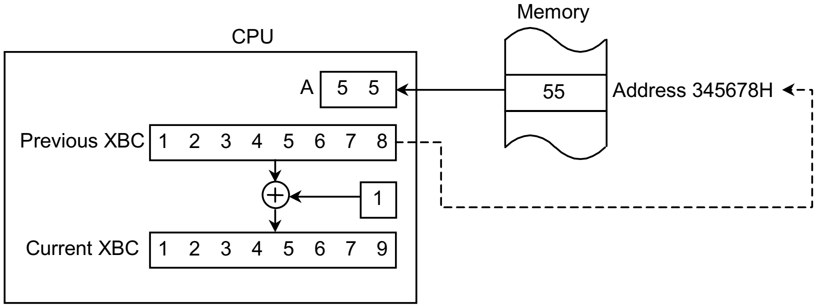
(5) Register Indirect Post-increment Addressing Mode

In this mode, the operand is the memory address specified by the contents of the register. After the operation, the contents of the register are incremented by the size of the operand.

Example 1: LD HL, (XIX+)



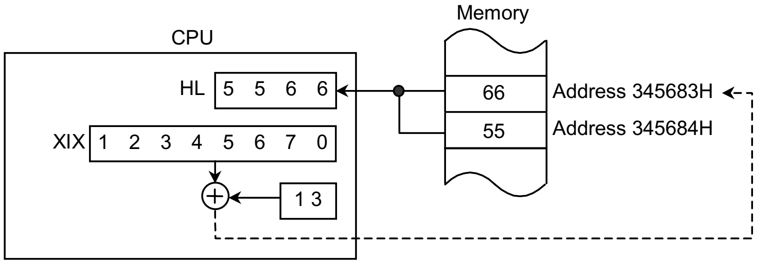
Example 2: LD A, (XBC+)



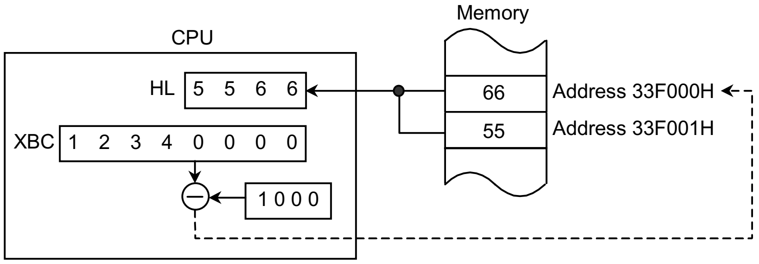
(6) Index Addressing Mode

In this mode, the operand is the memory address obtained by adding the contents of the specified register to the 8- or 16-bit displacement value in the instruction code.

Example 1: LD HL, (XIX + 13H)



Example 2: LD HL, (XBC – 1000H)

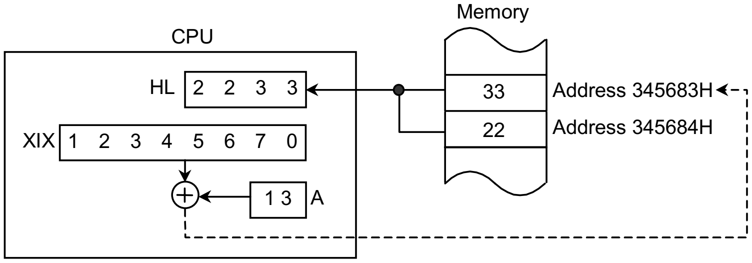


The displacement values range from −8000H to +7FFFH.

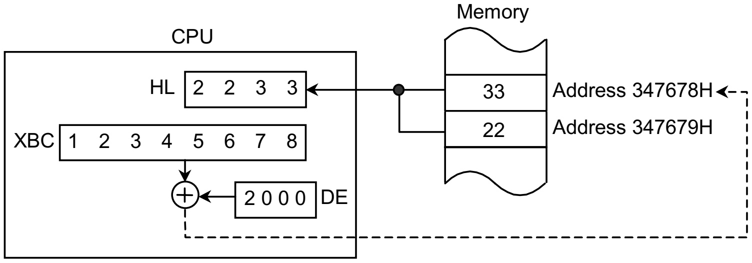
(7) Register Index Addressing Mode

In this mode, the operand is the memory address obtained by adding the contents of the register specified as the base to the register specified as the 8- or 16-bit displacement.

Example 1: LD HL, (XIX + A)



Example 2: LD HL, (XBC + DE)

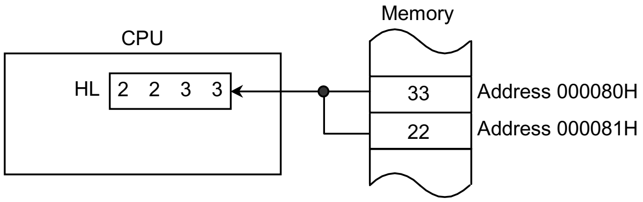


(8) Absolute Addressing Mode

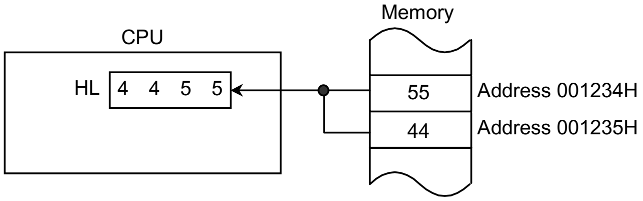
In this mode, the operand is the memory address specified by 1 to 3 bytes in the instruction code. Addresses 000000H to 0000FFH can be specified by 1 byte. Addresses 000000H to 00FFFFH can be specified by 2 bytes. Addresses 000000H to FFFFFFH can be specified by 3 bytes.

In this mode, addressing to 256-byte area (0H to FFH) which can be specified by 1 byte is called the direct addressing mode. In the direct addressing mode, a program memory area and execution time can be cut down.

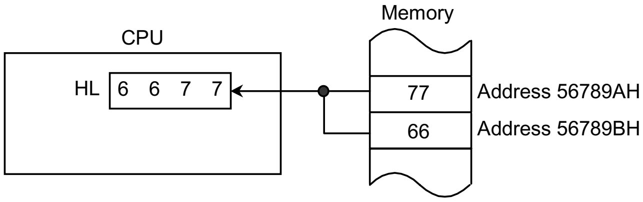
Example 1: LD HL, (80H)



Example 2: LD HL, (1234H)



Example 3: LD HL, (56789AH)



(9) Relative Addressing Mode

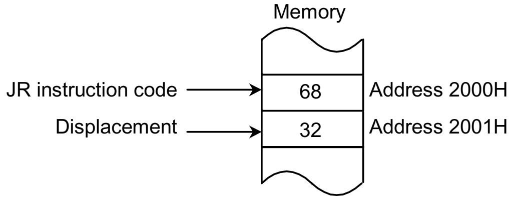
In this mode, the operand is the memory address obtained by adding the 8- or 16-bit displacement value to the address where the instruction code being executed is located.

In this mode, only the following five instructions can be used.

LDAR	R, $ + 4 + d16	
JR	cc, $ + 2 + d8	
JRL	cc, $ + 3 + d16	
CALR	$ + 3 + d16	
DJNZ	r, $ + 3 + d8	($: start address of instruction code)

In calculating the displacement object code value, the adjustment value (+2 to +4) depends on the instruction type.

Example 1: JR 2034H



In the above example, the displacement object code value is:

2034H – (2000H + 2) = 32H.

5. Instructions

In addition to its various addressing modes, the TLCS-900/L1 series also has a powerful instruction set. The basic instructions are classified into the following nine groups:

- Load instructions (8/16/32 bits)
- Exchange instructions (8/16 bits)
- Block transfer & Block search instructions (8/16 bits)
- Arithmetic operation instructions (8/16/32 bits)
- Logical operation instructions (8/16/32 bits)
- Bit operation instructions (1 bit)
- Special operations, CPU control instructions
- Rotate and Shift instructions (8/16/32 bits)
- Jump, Call, and Return instructions

Table 5.1 lists the basic instructions of the TLCS-900/L1 series. For details of instructions, see Appendix A; for the instruction list, Appendix B; for the instruction code map, Appendix C; and for the differences between the TLCS-90 and TLCS-900/L1 series, Appendix D.

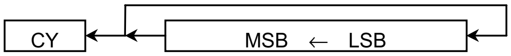
Table 5.1 TLCS-900/L1 Series basic instructions

LD	dst, src	Load dst ← src		
PUSH	src	Push src data to stack.		
		SP ← SP − size: (SP) ← src		
POP	dst	Pop data from stack to dst.		
		dst ← (SP): SP ← SP + size		
LDA	dst, src	Load address: set src effective address in dst.		
LDAR	dst, PC + dd	Load address relative:		
		set program counter relative address value in dst. dst ← PC + dd		
EX	dst1, dst2	Exchange dst1 and dst2 data.		
MIRR	dst	Mirror-invert dst bit pattern.		
LDI		Load increment		
LDIR		Load increment repeat		
LDD		Load decrement		
LDDR		Load decrement repeat		
CPI		Compare increment		
CPIR		Compare increment repeat		
CPD		Compare decrement		
CPDR		Compare decrement repeat		
ADD	dst, src	Add	dst ← dst + src	
ADC	dst, src	Add with carry	dst ← dst + src + CY	
SUB	dst, src	Subtract	dst ← dst − src	
SBC	dst, src	Subtract with carry	dst ← dst − src − CY	
CP	dst, src	Compare	dst − src	
AND	dst, src	And	dst ← dst	AND src
OR	dst, src	Or	dst ← dst	OR src
XOR	dst, src	Exclusive-or	dst ← dst	XOR src
INC	imm, dst	Increment	dst ← dst + imm	
DEC	imm, dst	Decrement	dst ← dst − imm	
MUL	dst, src	Multiply unsigned	dst ← dst (low) × src	
MULS	dst, src	Multiply signed	dst ← dst (low) × src	
DIV	dst, src	Divide unsigned		
		dst (low) ← dst ÷ src		
		dst (high) ← remainder		
		V flag set due to division by 0 or overflow.		
DIVS	dst, src	Divide signed		
		dst (low) ← dst ÷ src		
		dst (high) ← remainder: sign is same as that of dividend.		
		V flag set due to division by 0 or overflow.		

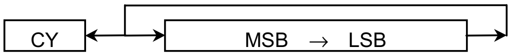
MULA	dst	Multiply and add $\underset{32\ \text{bit}}{\underline{\text{dst}}} \leftarrow \underset{32\ \text{bit}}{\underline{\text{dst}}} + (\underset{16\ \text{bit}}{\underline{\text{XDE}}}) \times (\underset{16\ \text{bit}}{\underline{\text{XHL}-}})$
MINC1	num, dst	Modulo increment 1
MINC2	num, dst	Modulo increment 2
MINC4	num, dst	Modulo increment 4
MDEC1	num, dst	Modulo decrement 1
MDEC2	num, dst	Modulo decrement 2
MDEC4	num, dst	Modulo decrement 4
NEG	dst	Negate dst ← 0 – dst (Twos complement)
CPL	dst	Complement dst ← not dst (Ones complement)
EXTZ	dst	Extend zero: set upper data of dst to 0.
EXTS	dst	Extend signed: copy the MSB of the lower data of dst to upper data.
DAA	dst	Decimal adjustment accumulator
PAA	dst	Pointer adjustment accumulator: when dst is odd, increment dst by 1 to make it even. if dst (0) = 1 then dst ← dst + 1.
LDCF	bit, src	Load carry flag: copy src<bit> value to C flag.
STCF	bit, dst	Store carry flag: copy C flag value to dst<bit>.
ANDCF	bit, src	And carry flag: and src<bit> value and C flag, then load the result to C flag.
ORCF	bit, src	Or carry flag: or src<bit> and C flag, then load result to C flag.
XORCF	bit, src	Exclusive-or carry flag: exclusive-or src<bit> value and C flag, then load result to C flag.
RCF		Reset carry flag: reset C flag to 0.
SCF		Set carry flag: set C flag to 1.
CCF		Complement carry flag: invert C flag value.
ZCF		Zero flag to carry flag: copy inverted value of Z flag to C flag.
BIT	bit, src	Bit test: Z flag ← not src<bit>
RES	bit, dst	Bit reset
SET	bit, dst	Bit set
CHG	bit, dst	Bit change dst<bit> ← not dst<bit>
TSET	bit, dst	Bit test and set: Z flag ← not dst<bit> dst<bit> ← 1

BS1F	A, dst	Bit search 1 forward: search dst for the first bit set to 1 starting from the LSB, then set the bit number in the A register.
BS1B	A,dst	Bit search 1 backward: search dst for the first bit set to 1 starting fom the MSB, then set the bit number in the A register.
NOP		No operation
EI	imm	Enable interrupt. IFF ← imm
DI		Disable maskable interrupt. IFF ← 7
PUSH	SR	Push status registers.
POP	SR	Pop status registers.
SWI	imm	Software interrupt PUSH PC&SR: JP 8000H + 10H × imm
HALT		Halt CPU.
LDC	CTRL-REG, reg	Load control: copy the register contents to control register of CPU.
LDC	reg, CTRL-REG	Load control: copy the control register contents to register.
LDX	dst, src	Load extract. dst ← src
LINK	reg, dd	Link: generate stack frame. PUSH reg LD reg, XSP ADD XSP, dd
UNLK	reg	Unlink: delete stack frame. LD XSP, reg POP reg
LDF	imm	Load register file pointer: specify register bank. RFP ← imm
INCF		Increment register file pointer: move to new register bank. RFP ← RFP + 1
DECF		Decrement register file pointer: return to previous register bank. RFP ← RFP − 1
SCC	cc, dst	Set dst with condition codes. if cc then dst ←1 else dst ← 0.

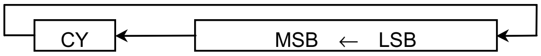
RLC num, dst | Rotate left without carry



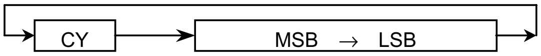
RRC num, dst | Rotate right without carry



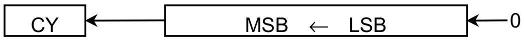
RL num, dst | Rotate left



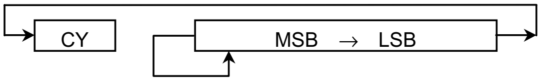
RR num, dst | Rotate right



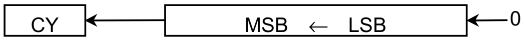
SLA num, dst | Shift left arithmetic



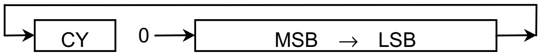
SRA num, dst | Shift right arithmetic



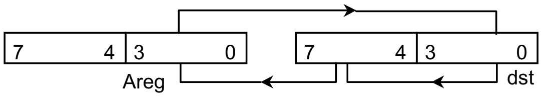
SLL num, dst | Shift left logical



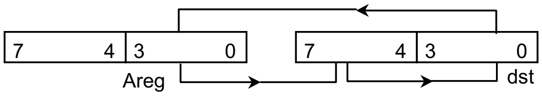
SRL num, dst | Shift right logical



RLD dst | Rotate left digit



RRD dst | Rotate right digit



JR cc, PC+d | Jump relative (8-bit displacement)
    if cc then PC ← PC + d.

JRL cc, PC + dd | Jump relative long (16-bit displacement)
    if cc then PC ← PC + dd.

JP cc, dst | Jump
    if cc then PC ← dst.

CALR RC + dd | Relative call (16-bit displacement)
    PUSH PC: PC ← PC + dd.

CALL cc, dst | Call relative
    if cc then PUSH PC: PC ← dst.

DJNZ dst, PC + d | Decrement and jump if non-zero
    dst ← dst – 1
    if dst ≠ 0 then PC ← PC + d.

RET cc | Return
    if cc then POP PC.

RETD dd | Return and deallocate
    RET
    XSP ← XSP + dd

RETI | Return from interrupt
    POP SR&PC

Table 5.2 Instruction list

BWL	LD	reg, reg	BWL	INC	imm3, reg	---	NOP	
BWL	LD	reg, imm		DEC	imm3, mem.B/W			
BWL	LD	reg, mem						
BWL	LD	mem, reg				---	EI	[imm3]
BW-	LD	mem, imm				---	DI	
BW-	LD	(nn), mem	BW-	MUL	reg, reg	-W-	*PUSH	SR
BW-	LD	mem, (nn)		*MULS	reg, imm	-W-	*POP	SR
				DIV	reg, mem	---	SWI	[imm3]
				*DIVS		---	HALT	
BWL	PUSH	reg/F				BWL	*LDC	CTRL – R, reg
BW-	PUSH	imm	-W-	*MULA	reg	BWL	*LDC	reg, CTRL – R
BW-	PUSH	mem				B--	*LDX	(n), n
			-W-	*MINC1	imm, reg			
BWL	POP	reg/F	-W-	*MINC2	imm, reg	--L	*LINK	reg, dd
BW-	POP	mem	-W-	*MINC4	imm, reg	--L	*UNLK	reg
			-W-	*MDEC1	imm, reg	---	*LDF	imm3
			-W-	*MDEC2	imm, reg	---	*INCF	
-WL	LDA	reg, mem	-W-	*MDEC4	imm, reg	---	*DECF	
-WL	LDAR	reg, PC+dd				BW-	*SCC	cc, reg
			BW-	NEG	reg			
			BW-	CPL	reg	BWL	RLC	imm, reg
			-WL	*EXTZ	reg		RRC	A, reg
B--	EX	F, F’	-WL	*EXTS	reg		RL	mem. B/W
BW-	EX	reg, reg	B--	DAA	reg		RR	
BW-	EX	mem, reg	-WL	*PAA	reg		SLA	
							SRA	
							SLL	
-W-	*MIRR	reg	BW-	*LDCF	imm, reg		SRL	
				*STCF	A, reg			
				*ANDCF	imm, mem.B	B--	RLD	[A,] mem
				*ORCF	A, mem.B	B--	RRD	[A,] mem
BW-	LDI			*XORCF				
BW-	LDIR							
BW-	LDD		---	RCF		---	JR	[cc,] PC + d
BW-	LDDR		---	SCF		---	JRL	[cc,] PC + dd
			---	CCF		---	JP	[cc,] mem
			---	*ZCF		---	CALR	PC + dd
BW-	CPI					---	CALL	[cc,] mem
BW-	CPIR		BW-	BIT	imm, reg			
BW-	CPD			RES	imm, mem.B	BW-	DJNZ	[reg], PC + d
BW-	CPDR			SET				
				*CHG		---	RET	[cc]
				TSET		---	*RETD	dd
BWL	ADD	reg, reg				---	RETI	
	ADC	reg, imm	-W-	*BS1F	A, reg			
	SUB	reg, mem		*BS1B				
	SBC	mem, reg						
	CP	mem, imm.B/W						
	AND							
	OR							
	XOR							

B = Byte (8 bit), W = Word (16 bit), L = Long – Word (32 bit).

* : Indicates instruction added to the TLCS-90 series.

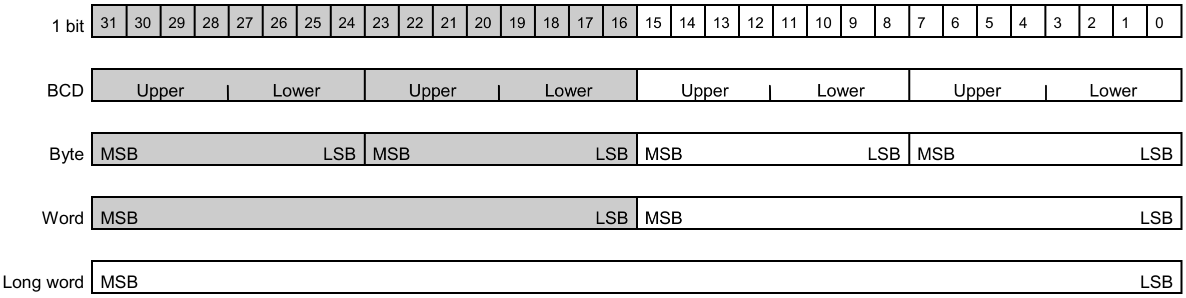
[ ] : Indicates can be omitted.

6. Data Formats

The TLCS-900/L1 series can handle 1/4/8/16/32-bit data.

(1) Register Data Format

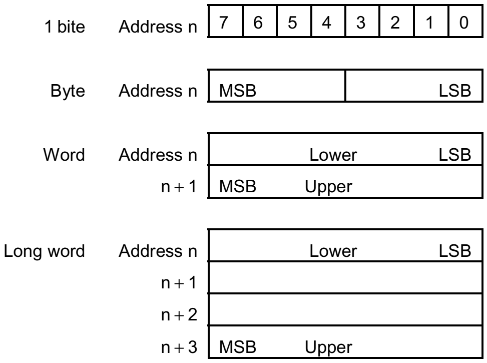
<Data image>



Note 1: To access the parts indicated by ▨, the instruction code is one byte longer than when accessing the other parts.

(2) Memory Data Format

<Data image>



Note 2: There are no restrictions on the location of word or long word data in memory. They can be located from even or odd numbered address.

Note 3: When the PUSH instruction is used to save data to the stack area, the stack pointer is decremented, then the data is saved.

Example: PUSH HL; XSP ← XSP − 2
(XSP) ← L
(XSP + 1) ← H

This is the same in register indirect pre-decrement mode. The order is reversed in the TLCS-90 series: data is saved first, then the stack pointer is decremented.

Example: PUSH HL; (XSP − 1) ← H
(XSP − 2) ← L
XSP ← XSP − 2

(3) Dynamic Bus Sizing

The TLCS-900/L1 series can switch between 8- and 16-bit data buses dynamically during each bus cycle. This is called dynamic bus sizing. The function enables external memory extension using both 8- and 16-bit data bus memories. Products with a built-in chip select/wait controller can control external data bus size for each address area.

Table 6.1 Dynamic Bus Sizing

Operand data size	Operand start address	Data size at memory side	CPU address	CPU data	
				D15 to D8	D7 to D0
8 bits	2n + 0	8 bits	2n + 0	xxxxx	b7 to b0
	(even)	16 bits	2n + 0	xxxxx	b7 to b0
	2n + 1	8 bits	2n + 1	xxxxx	b7 to b0
	(odd)	16 bits	2n + 1	b7 to b0	xxxxx
16 bits	2n + 0	8 bits	2n + 0	xxxxx	b7 to b0
	(even)		2n + 1	xxxxx	b15 to b8
		16 bits	2n + 0	b15 to b8	b7 to b0
	2n + 1	8 bits	2n + 1	xxxxx	b7 to b0
	(odd)		2n + 2	xxxxx	b15 to b8
		16 bits	2n + 1	b7 to b0	xxxx
			2n + 2	xxxxx	b15 to b8
32 bits	2n + 0	8 bits	2n + 0	xxxxx	b7 to b0
	(even)		2n + 1	xxxxx	b15 to b8
			2n + 2	xxxxx	b23 to b16
			2n + 3	xxxxx	b31 to b24
		16 bits	2n + 0	b15 to b8	b7 to b0
			2n + 2	b31 to b24	b23 to b16
	2n + 1	8 bits	2n + 1	xxxxx	b7 to b0
	(odd)		2n + 2	xxxxx	b15 to b8
			2n + 3	xxxxx	b23 to b16
			2n + 4	xxxxx	b31 to b24
		16 bits	2n + 1	b7 to b0	xxxxx
			2n + 2	b23 to b16	b15 to b8
			2n + 4	xxxx	b31 to b24

xxxxx: During read, indicates the data input to the bus are ignored. During write, indicates the bus is at high impedance and the write strobe signal is non-active.

(4) Internal Data Bus Format

With the TLCS-900/L1 series, the CPU and the internal memory (built-in ROM or RAM) are connected via a 16-bit internal data bus. The internal memory operates with 0 wait. The CPU and the built-in I/Os are connected using an 8-bit internal data bus. This is because the built-in I/O access speed has little influence on the overall system operation speed.

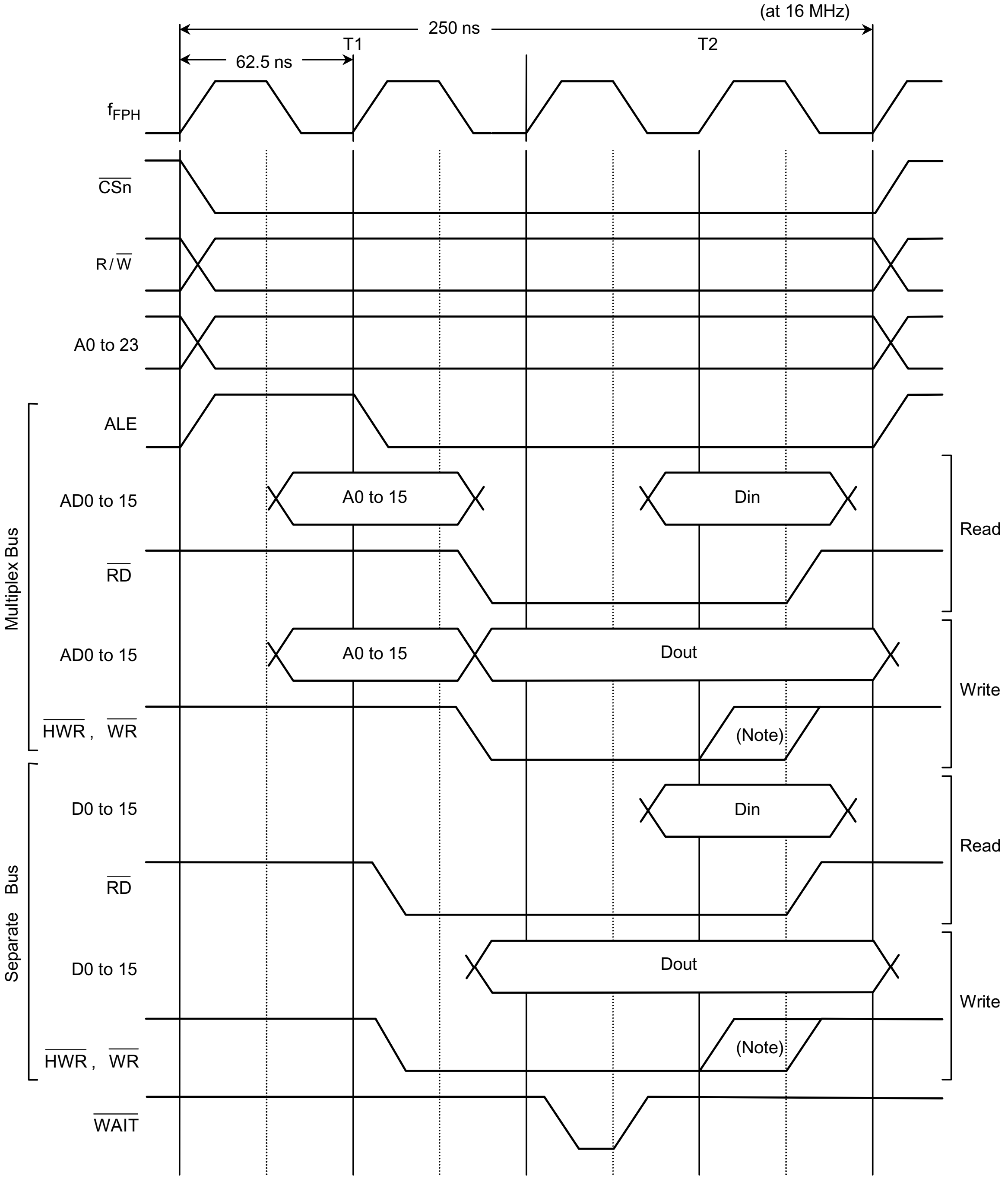
Overall system operation speed depends largely on the speed of program memory access.

7. Basic Timings

The TLCS-900/L1 series runs the following basic timings.

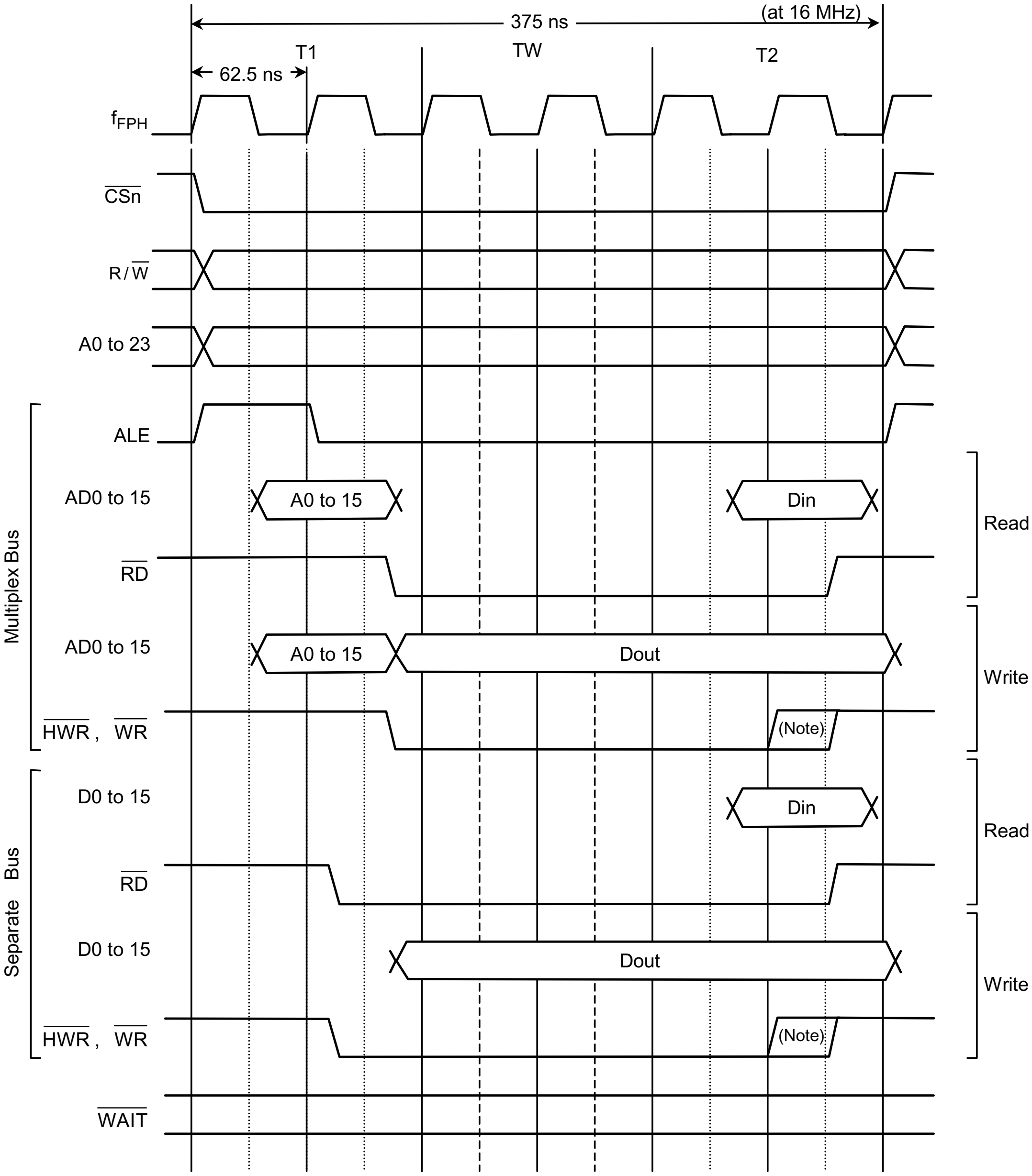
- Read cycle
- Write cycle
- Dummy cycle
- Interrupt receive timing
- Reset

Figure 7.1 to Figure 7.8 show the basic timings.



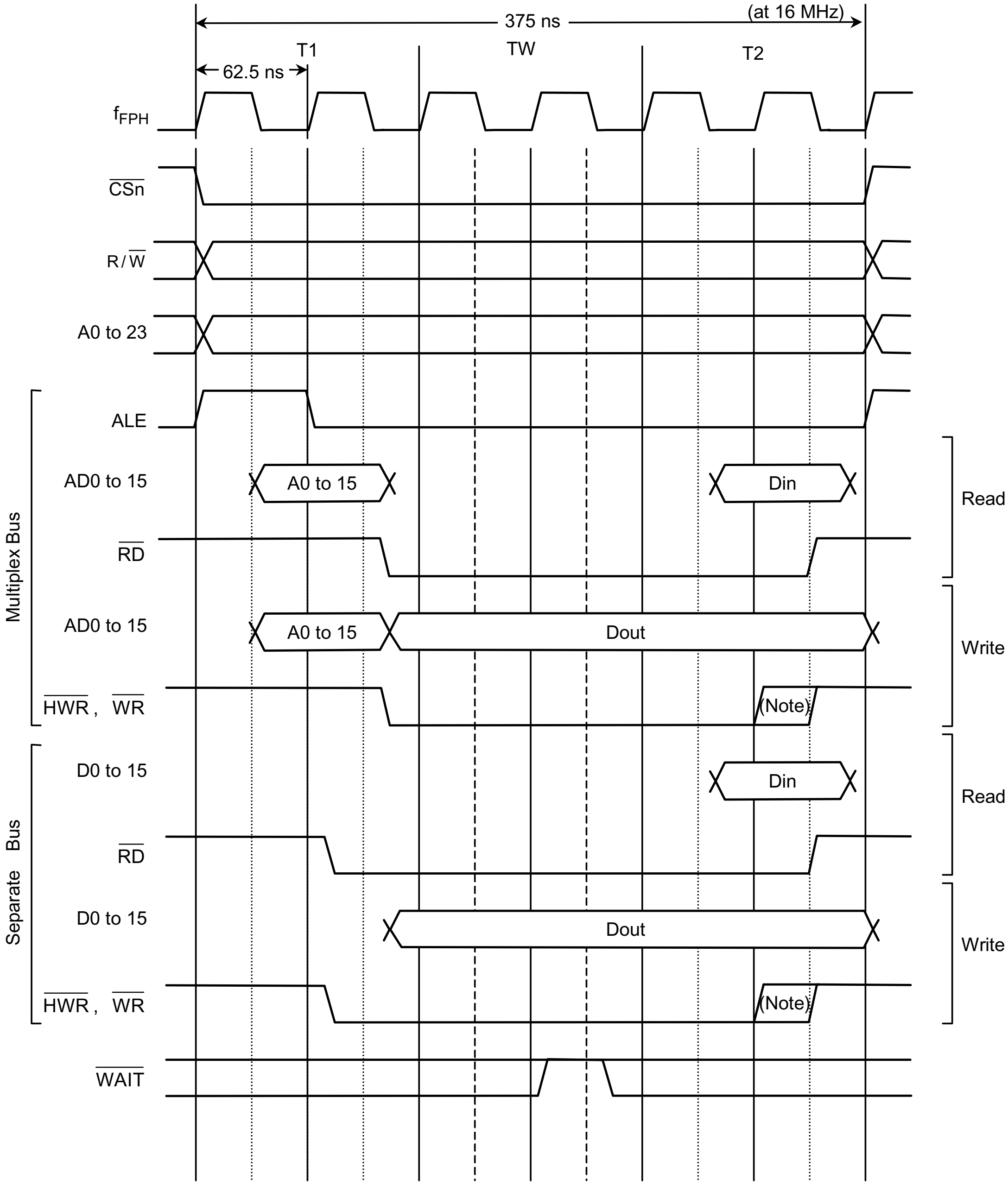
Note: $\overline{HWR}$ and $\overline{WR}$ timing depends on product.

Figure 7.1 0 Wait Read/Write cycle



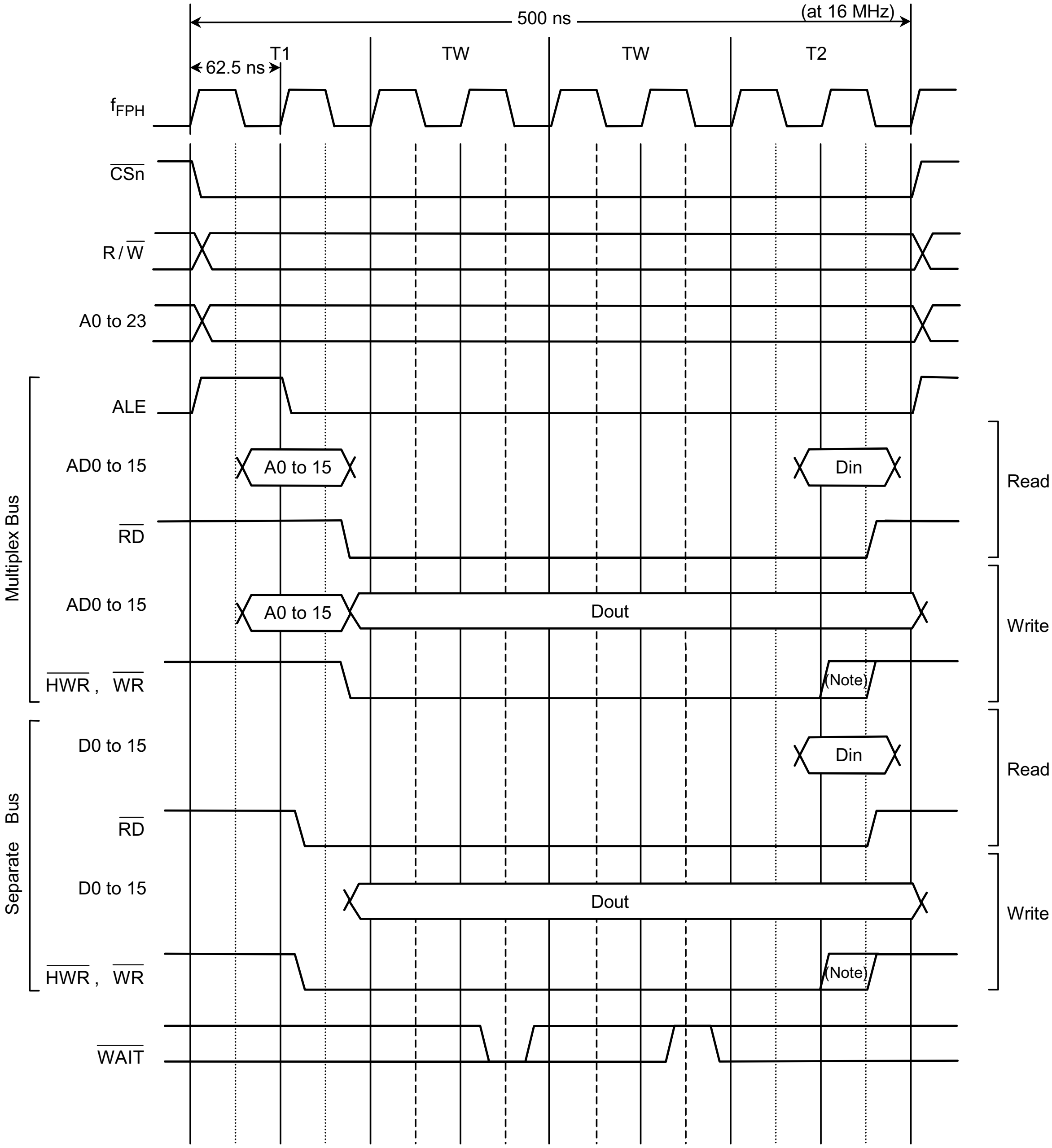
Note: $\overline{HWR}$ and $\overline{WR}$ timing depends on product.

Figure 7.2 1 Wait Read/Write cycle



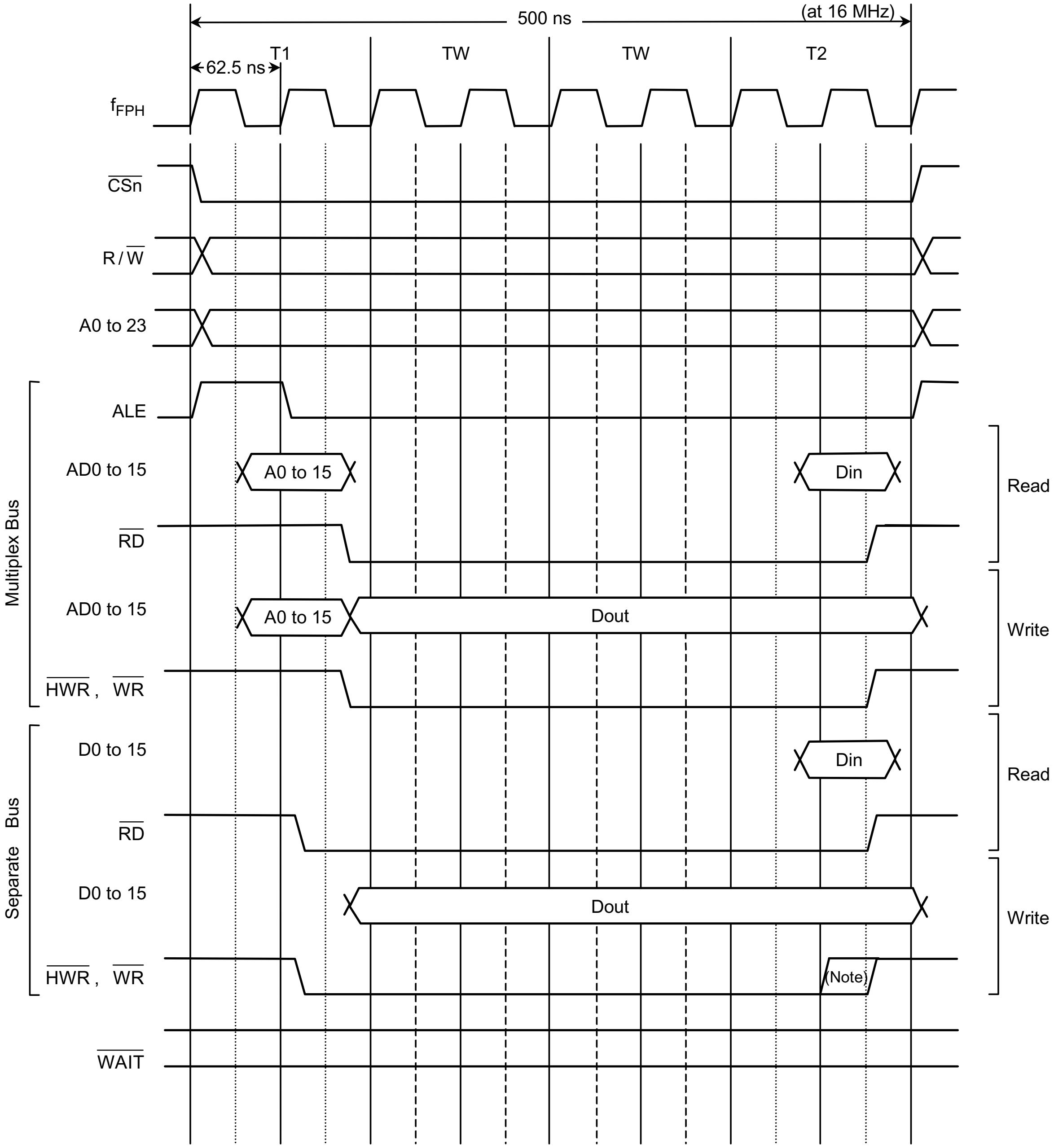
Note: HWR and WR timing depends on product.

Figure 7.3 1 Wait + n Read/Write cycle (n = 0)



Note: HWR and WR timing depends on product.

Figure 7.4 1 Wait + n Read/Write cycle (n = 1)



Note: $\overline{\text{HWR}}$ and $\overline{\text{WR}}$ timing depends on product.

Figure 7.5 2 Wait Read/Write cycle

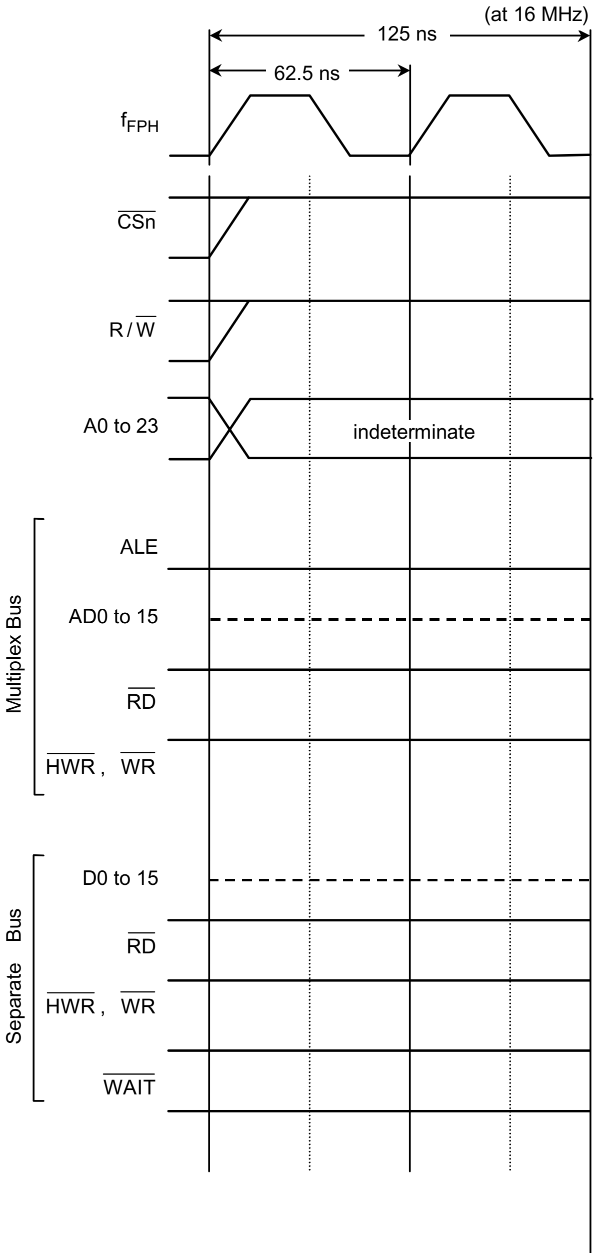
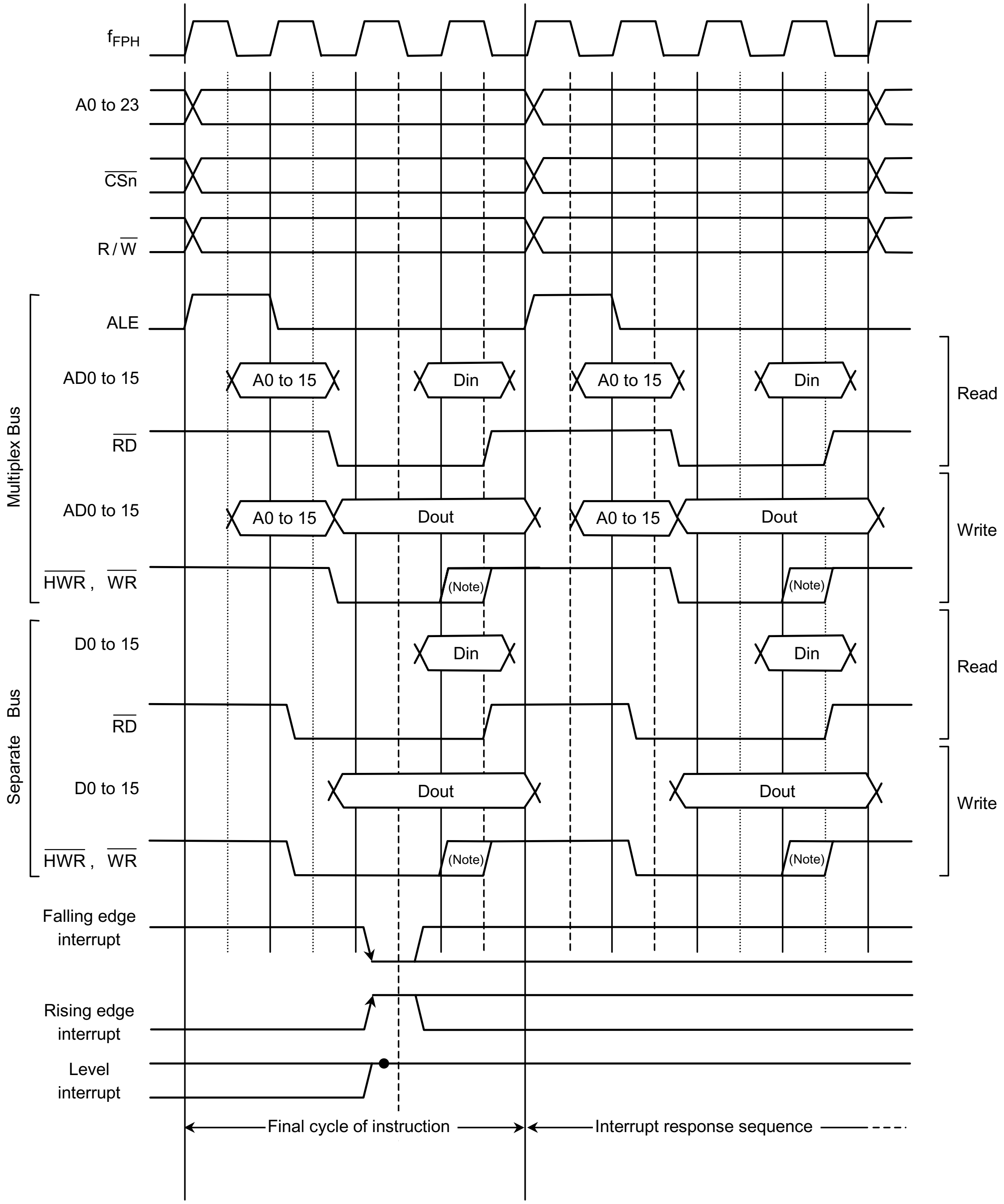


Figure 7.6 1-state dummy cycle



Note : This timing chart is a theoretical example. In practice, due to the operation of the bus interface unit in the CPU, external bus and internal interrupt receive timings do not correspond one to one. $\overline{\text{HWR}}$ and $\overline{\text{WR}}$ timings depend on product.

Figure 7.7 Interrupt receive timing

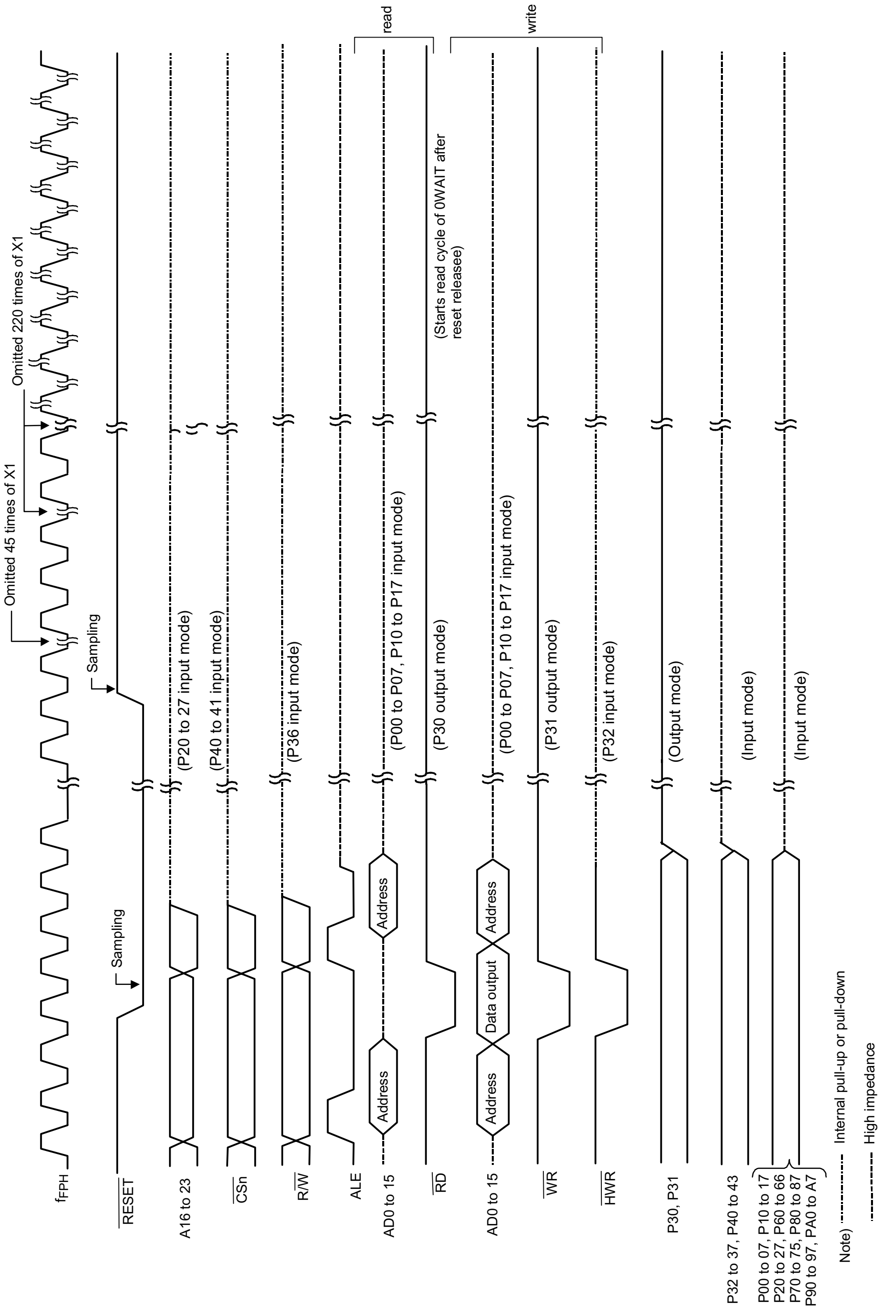


Figure 7.8 Reset timings (internal ROM operation: e.g. TMP91CW12)

Appendix A: Details of Instructions

- Instruction List

1. Load
   LD PUSH POP LDA LDAR

2. Exchange
   EX MIRR

3. Load Increment/Decrement & Compare Increment/Decrement
   LDI LDIR LDD LDDR CPI CPIR CPD CPDR

4. Arithmetic operations
   ADD ADC SUB SBC CP INC DE NEG
   EXTZ EXTS DAA PAA MUL MULS DIV DIVS
   MULA MINC MDEC

5. Logical operations
   AND OR XOR CPL

6. Bit operations
   LDCF STCF ANDCF ORCF XORCF RCF SCF CCF
   ZCF BIT RES SET CHG TSET BS1

7. Special operations and CPU control
   NOP EI DI PUSH–SR POP–SR SWI HALT LDC
   LDX LINK UNLK LDF INCF DECF SCC

8. Rotate and shift
   RLC RRC RL RR SLA SRA SLL SRL
   RLD RRD

9. Jump, call, and return
   JP JR JRL CALL CALR DJNZ RET RETD
   RETI

- Explanations of symbols used in this document

dst	Destination: destination of data transfer or operation result load.	
src	Source: source of data transfer or operation data read.	
num	Number: numerical value.	
condition	Condition: based on flag status.	
R	Eight general–purpose registers including 8/16/32-bit current bank registers.	
	8-bit registers : W, A, B, C, D, E, H, L	(only eight registers)
	16-bit registers : WA, BC, DE, HL, IX, IY, IZ, SP	(only eight registers)
	32-bit registers : XWA, XBC, XDE, XHL, XIX, XIY, XIZ, XSP	(only eight registers)
r	8/16/32-bit general-purpose registers	(Please refer to Register map on page CPU900L1-45.)
r16	16-bit general-purpose registers	
r32	32-bit general-purpose registers	
cr	All 8/16/32-bit CPU control registers DMAS0 to 3, DMAD0 to 3, DMAC0 to 3, DMAM0 to 3, INTNEST	
A	A register (8 bits)	
F	Flag registers (8 bits)	
F’	Inverse flag registers (8 bits)	
SR	Status registers (16 bits)	
PC	Program counter (in minimum mode, 16 bits; in maximum mode, 32 bits)	
(mem)	8/16/32-bit memory data	
mem	Effective address value	
<W>	When the operand size is a word, W must be specified.	
[ ]	Operands enclosed in square brackets can be omitted.	
#	8/16/32-bit immediate data.	
#3	3-bit immediate data : 0 to 7 or 1 to 8 ... for abbreviated codes.	
#4	4-bit immediate data : 0 to 15 or 1 to 16	
d8	8-bit displacement : –80H to + 7FH	
d16	16-bit displacement : –8000H to + 7FFFH	
cc	Condition code	
CY	Carry flag	
Z	Zero flag	
(#8)	Direct addressing: (00H) to (0FFH) ... 256–byte area	
(#16)	64K–byte area addressing: (0000H) to (0FFFFH)	
(–r32)	Pre–decrement addressing	
(r32+)	Post–increment addressing	
$	Start address of instruction	

- Explanations of symbols in object codes

z
zz
zzz
s

Operand size specify code

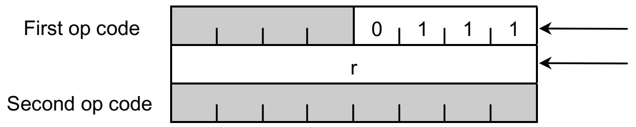
	Byte	Word	Long word
z	0	1	–
zz	00	01	10
zzz	010	011	100
s	–	0	1

R
r

Register specify code

Code	Byte	Word	Long word
000	W	WA	XWA
001	A	BC	XBC
010	B	DE	XDE
011	C	HL	XHL
100	D	IX	XIX
101	E	IY	XIY
110	H	IZ	XIZ
111	L	SP	XSP

Note: In addition to the above, all registers can be specified by r using extension codes. In this case, the number of execution states increases by 1. The format is shown below.



Sets the lower 4 bits to 0111.

Inserts the register code specified by 8 bits between the first and second op codes.

The code value in r must be:

Multiple of 2, if accessed as a word register.

Multiple of 4, if accessed as a long word.

For registers specified by 8 bits, see Register Maps.

mem

Memory addressing mode specify code

(XWA)	=	– 0 – – 0000			
(XBC)	=	– 0 – – 0001			
(XDE)	=	– 0 – – 0010			
(XHL)	=	– 0 – – 0011			
(XIX)	=	– 0 – – 0100			
(XIY)	=	– 0 – – 0101			
(XIZ)	=	– 0 – – 0110			
(XSP)	=	– 0 – – 0111			
(XWA+d8)	=	– 0 – – 1000	d<7:0>		
(XBC+d8)	=	– 0 – – 1001	d<7:0>		
(XDE+d8)	=	– 0 – – 1010	d<7:0>		
(XHL+d8)	=	– 0 – – 1011	d<7:0>		
(XIX+d8)	=	– 0 – – 1100	d<7:0>		
(XIY+d8)	=	– 0 – – 1101	d<7:0>		
(XIZ+d8)	=	– 0 – – 1110	d<7:0>		
(XSP+d8)	=	– 0 – – 1111	d<7:0>		
(#8)	=	– 1 – – 0000	#<7:0>		
(#16)	=	– 1 – – 0001	#<7:0>	#<15:8>	
(#24)	=	– 1 – – 0010	#<7:0>	#<15:8>	#<23:16>
(r32)	=	– 1 – – 0011	r32’ 00		
(r32+d16)	=	– 1 – – 0011	r32’ 01	d<7:0>	d<15:8>
(r32+r8)	=	– 1 – – 0011	000000 11	r32’	r8
(r32+r16)	=	– 1 – – 0011	000001 11	r32’	r16
(−r32)	=	– 1 – – 0100	r32’ zz		
(r32+)	=	– 1 – – 0101	r32’ zz		

<7:0> = Indicates the data bit range. This example means 8-bit data from bit 0 to bit 7.

r32: 32-bit register
r16: Signed 16-bit register
r8: Signed 8-bit register

Zz = Code used to specify the value of increments or decrements.
00: ±1
01: ±2
10: ±4
11: (Not defined)

r32’ = Upper 6 bits of register code

cc

Condition codes

Code	Symbol	Description	Conditional expression
0000	F	always False	–
1000	（none）	always True	–
0110	Z	Zero	Z = 1
1110	NZ	Not Zero	Z = 0
0111	C	Carry	C = 1
1111	NC	Not Carry	C = 0
1101	PL or P	PLus	S = 0
0101	MI or M	MInus	S = 1
1110	NE	Not Equal	Z = 0
0110	EQ	EQual	Z = 1
0100	OV	OVerflow	P/V = 1
1100	NOV	No OVerflow	P/V = 0
0100	PE	Parity is Even	P/V = 1
1100	PO	Parity is Odd	P/V = 0
1001	GE	Greater than or Equal (signed)	(S xor P/V) = 0
0001	LT	Less Than (signed)	(S xor P/V) = 1
1010	GT	Greater Than (signed)	[Z or (S xor P/V)] = 0
0010	LE	Less than or Equal (signed)	[Z or (S xor P/V)] = 1
1111	UGE	Unsigned Greater than or Equal	C = 0
0111	ULT	Unsigned Less Than	C = 1
1011	UGT	Unsigned Greater Than	(C or Z) = 0
0011	ULE	Unsigned Less than or Equal	(C or Z) = 1

- Register map "r" (MAX mode)

	+3	+2	+1	+0	
00H	QW0 (QWA	0) QA0 <XWA	0> RW0 (RWA	0) RA0	Bank 0
04H	QB0 (QBC	0) QC0 <XBC	0> RB0 (RBC	0) RC0	
08H	QD0 (QDE	0) QE0 <XDE	0> RD0 (RDE	0) RE0	
0CH	QH0 (QHL	0) QL0 <XHL	0> RH0 (RHL	0) RL0	
10H	QW1 (QWA	1) QA1 <XWA	1> RW1 (RWA	1) RA1	Bank 1
14H	QB1 (QBC	1) QC1 <XBC	1> RB1 (RBC	1) RC1	
18H	QD1 (QDE	1) QE1 <XDE	1> RD1 (RDE	1) RE1	
1CH	QH1 (QHL	1) QL1 <XHL	1> RH1 (RHL	1) RL1	
20H	QW2 (QWA	2) QA2 <XWA	2> RW2 (RWA	2) RA2	Bank 2
24H	QB2 (QBC	2) QC2 <XBC	2> RB2 (RBC	2) RC2	
28H	QD2 (QDE	2) QE2 <XDE	2> RD2 (RDE	2) RE2	
2CH	QH2 (QHL	2) QL2 <XHL	2> RH2 (RHL	2) RL2	
30H	QW3 (QWA	3) QA3 <XWA	3> RW3 (RWA	3) RA3	Bank 3
34H	QB3 (QBC	3) QC3 <XBC	3> RB3 (RBC	3) RC3	
38H	QD3 (QDE	3) QE3 <XDE	3> RD3 (RDE	3) RE3	
3CH	QH3 (QHL	3) QL3 <XHL	3> RH3 (RHL	3) RL3	
D0H	QW' (Q	WA') QA' <X	WA'> W' (W	A') A'	Previous bank
D4H	QB' (Q	BC') QC' <X	BC'> B' (B	C') C'	
D8H	QD' (Q	DE') QE' <X	DE'> D' (D	E') E'	
DCH	QH' (Q	HL') QL' <X	HL'> H' (H	L') L'	
E0H	QW (Q	WA) QA <X	WA > W (W	A) A	Current bank
E4H	QB (Q	BC) QC <X	BC > B (B	C) C	
E8H	QD (Q	DE) QE <X	DE > D (D	E) E	
ECH	QH (Q	HL) QL <X	HL > H (H	L) L	
F0H	QIXH (Q	IX) QIXL <X	IX> IXH (I	X) IXL	
F4H	QIYH (Q	IY) QIYL <X	IY> IYH (I	Y) IYL	
F8H	QIZH (Q	IZ) QIZL <X	IZ> IZH (I	Z) IZL	
FCH	QSPH (Q	SP) QSPL <X	SP> SPH (S	P) SPL	

( ): Word register name (16 bits)

< >: Long word register name (32 bits)

- Control register map cr

	+3	+2	+1	+0	
00H		<DMA	SO>		micro DMA source register
04H		<DMA	S1>		
08H		<DMA	S2>		
0CH		<DMA	S3>		
10H		<DMA	D0>		micro DMA destination register
14H		<DMA	D1>		
18H		<DMA	D2>		
1CH		<DMA	D3>		
20H		DMAM0	(DMA	C0)	micro DMA mode/counter register
24H		DMAM1	(DMA	C1)	
28H		DMAM2	(DMA	C2)	
2CH		DMAM3	(DMA	C3)	
3CH			(INT	NEST)	Interrupt Nesting Counter

( ): Word register name (16 bits)

< >: Long word register name (32 bits)

ADC dst, src

<Add with Carry>

Operation: dst ← dst + src + CY

Description: Adds the contents of dst, src, and carry flag, and transfers the result to dst.

Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
○	○	○	ADC	R, r	1 1 z z 1 r 1 0 0 1 0 R
○	○	○	ADC	r, #	1 1 z z 1 r 1 1 0 0 1 0 0 1 #<7:0> #<15:8> #<23:16> #<31:24>
○	○	○	ADC	R, (mem)	1 m z z m m m m 1 0 0 1 0 R
○	○	○	ADC	(mem), R	1 m z z m m m m 1 0 0 1 1 R
○	○	×	ADC<W>	(mem), #	1 m 0 z m m m m 0 0 1 1 1 0 0 1 #<7:0> #<15:8>

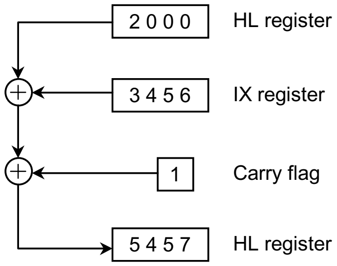
Flags:

S	Z	H	V	N	C
*	*	*	*	0	*

S = MSB value of the result is set.
Z = 1 is set if the result is 0, otherwise 0.
H = 1 is set if a carry from bit 3 to bit 4 occurs as a result of the operation; otherwise, 0. If the operand is 32-bit, an undefined value is set.
V = 1 is set if an overflow occurs as a result of the operation; otherwise, 0.
N = Cleared to zero.
C = 1 is set if a carry occurs from the MSB, otherwise 0.

Execution example: ADC HL,IX

When the HL register = 2000H, the IX register = 3456H, and the carry flag = 1, execution sets the HL register to 5457H.



ADD dst, src

<Add>

Operation: dst ← dst + src

Description: Adds the contents of dst to those of src and transfers the result to dst.

Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
○	○	○	ADD	R, r	1 1 | z z | 1 | r
					1 0 0 0 0 | R
○	○	○	ADD	r, #	1 1 | z z | 1 | r
					1 1 0 0 1 0 0 0
					#<7:0>
					#<15:8>
					#<23:16>
					#<31:24>
○	○	○	ADD	R, (mem)	1 | m | z z | m m m m
					1 0 0 0 0 | R
○	○	○	ADD	(mem), R	1 | m | z z | m m m m
					1 0 0 0 1 | R
○	○	×	ADD<W>	(mem), #	1 | m | 0 | z | m m m m
					0 0 1 1 1 0 0 0
					#<7:0>
					#<15:8>

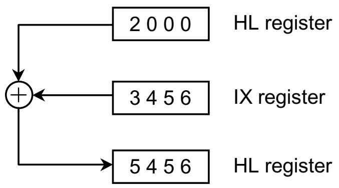
Flags:

S	Z	H	V	N	C
*	*	*	*	0	*

S = MSB value of the result is set.
Z = 1 is set if the result is 0, otherwise 0.
H = 1 is set if a carry from bit 3 to bit 4 occurs as a result of the operation, otherwise 0. If the operand is 32-bit, an undefined value is set.
V = 1 is set if an overflow occurs as a result of the operation, otherwise 0.
N = Cleared to zero.
C = 1 is set if a carry occurs from the MSB, otherwise 0.

Execution example: ADD HL,IX

When the HL register = 2000H and the IX register = 3456H, execution sets the HL register to 5456H.



AND dst, src

<And>

Operation: dst ← dst AND src

Description: Ands the contents of dst and src, then transfers the result to dst.

(Truth table)

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
○	○	○	AND	R, r	1 1 z z 1 r 1 1 0 0 0 R
○	○	○	AND	r, #	1 1 z z 1 r 1 1 0 0 1 1 0 0 #<7:0> #<15:8> #<23:16> #<31:24>
○	○	○	AND	R, (mem)	1 m z z m m m m 1 1 0 0 0 R
○	○	○	AND	(mem), R	1 m z z m m m m 1 1 0 0 1 R
○	○	×	AND<W>	(mem), #	1 m 0 z m m m m 0 0 1 1 1 1 0 0 #<7:0> #<15:8>

Flags:

S	Z	H	V	N	C
*	*	1	*	0	0

S = MSB value of the result is set.
Z = 1 is set if the result is 0, otherwise 0.
H = 1 is set.
V = 1 is set if a parity of the result is even, 0 if odd. If the operand is 32 bits, an undefined value is set.
N = Cleared to zero.
C = Cleared to zero.

Execution example: AND HL,IX
When the HL register = 7350H and the IX register = 3456H, execution sets the HL register to 3050H.

```
       0111  0011  0101  0000 ←  HL register (before execution)
AND)   0011  0100  0101  0110 ←  IX register  (before execution)
─────────────────────────────
       0011  0000  0101  0000 ←  HL register (after execution)
```

ANDCF num, src

<And Carry Flag>

Operation: CY ← CY AND src<num>

Description: Ands the contents of the carry flag and bit num of src, and transfers the result to the carry flag.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	×	ANDCF	#4, r	1 1 0 z 1 r 0 0 1 0 0 0 0 0 0 0 0 0 # 4
○	○	×	ANDCF	A, r	1 1 0 z 1 r 0 0 1 0 1 0 0 0
○	×	×	ANDCF	#3, (mem)	1 m 1 1 m m m m 1 0 0 0 0 #3
○	×	×	ANDCF	A, (mem)	1 m 1 1 m m m m 0 0 1 0 1 0 0 0

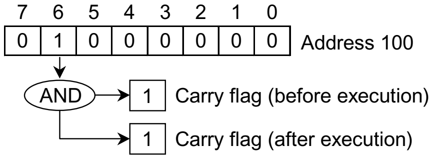
Notes: When bit num is specified by the A register, the value of the lower 4 bits of the A register is used as bit num. When the operand is a byte and the value of the lower 4 bits of bit num is from 8 to 15, the result is undefined.

Flags:

S	Z	H	V	N	C
–	–	–	–	–	*

S = No change
Z = No change
H = No change
V = No change
N = No change
C = The value obtained by anding the contents of the carry flag and the bit num of src is set.

Execution example: ANDCF 6,(100H)
When the contents of memory address 100 = 01000000B (binary) and the carry flag = 1, execution sets the carry flag to 1.



BIT num, src

<Bit test>

Operation: Z flag ← inverted value of src<num>

Description: Transfers the inverted value of the bit num of src to the Z flag.

Details:

Size Byte	 Word	 Long word	Mnemonic		Code
○	○	×	BIT	#4, r	1 1 0 z 1 r 0 0 1 1 0 0 1 1 0 0 0 0 # 4
○	×	×	BIT	#3, (mem)	1 m 1 1 m m m m 1 1 0 0 1 #3

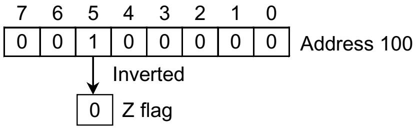
Flags:

S	Z	H	V	N	C
×	*	1	×	0	–

S = An undefined value is set.
Z = The inverted value of src <num> is set.
H = 1 is set.
V = An undefined value is set.
N = Reset to 0.
C = No change

Execution example: BIT 5, (100H)

When the contents of memory address 100 = 00100000B (binary), execution sets the Z flag to 0.



BS1B dst, src

<Bit Search 1 Backward>

Operation: dst ← src backward searched value

Description: Searches the src bit pattern backward (from MSB to LSB) for the first bit set to 1 and transfers the bit number to dst.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
×	○	×	BS1B	A, r	1 1 0 1 1 r 0 0 0 0 1 1 1 1

Note: dst in the operand must be the A register; src must be the register in words. If no bit set to 1 is found in the searched bit pattern, sets the A register to an undefined value and the V flag to 1.

Flags:

S	Z	H	V	N	C
–	–	–	*	–	–

S = No change
Z = No change
H = No change
V = 1 is set if the contents of src are all 0s (no bit is set to 1), otherwise 0.
N = No change
C = No change

Execution example: BS1B A,IX
When the IX register = 1200H, execution sets the A register to 0CH.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	IX register

Serch for 1.

BS1F dst, src

<Bit Search 1 Forward>

Operation: dst ← src forward searched result

Description: Searches the src bit pattern forward (from LSB to MSB) for the first bit set to 1 and transfers the bit number to dst.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
×	○	×	BS1F	A, r	1 1 0 1 1 r
					0 0 0 0 1 1 1 0

Note: dst in the operand must be the A register; src must be a register in words. If no bit set to 1 is found in the searched bit pattern, sets the A register to an undefined value and the V flag to 1.

Flags:

S	Z	H	V	N	C
–	–	–	*	–	–

S = No change
Z = No change
H = No change
V = 1 is set if the contents of src are all 0s (no bit is set to 1), otherwise 0.
N = No change
C = No change

Execution example: BS1F A,IX
When the IX register = 1200H, execution sets the A register to 09H.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	IX register

Serch for 1.

CALL condition, dst

<Call subroutine>

Operation: If cc is true, then XSP ← XSP – 4, (XSP) ← 32-bit PC, PC ← dst.

Description: If the operand condition is true, saves the contents of the program counter to the stack area and jumps to the program address specified by dst.

Details:

Mnemonic	Code							
CALL #16	0	0	0	1	1	1	0	0
	#<7:0>							
	#<15:8>							
CALL #24	0	0	0	1	1	1	0	1
	#<7:0>							
	#<15:8>							
	#<23:16>							
CALL [cc,] mem	1	m	1	1	m	m	m	m
	1	1	1	0		c	c	

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change

Z = No change

H = No change

V = No change

N = No change

C = No change

Execution example: CALL 9000H

When the stack pointer XSP is 100H, executing this instruction at memory address 8000H writes the return address 8003H (long word data) to memory address 0FCH, sets the stack pointer XSP to 0FCH, and jumps to address 9000H.

CALR dst

<Call Relative>

Operation: XSP ← XSP − 4, (XSP) ← 32-bit PC,PC ← dst.

Description: Saves the contents of the program counter to the stack area and makes a relative jump to the program address specified by dst.

Details:

Mnemonic	Code
CALR $ + 3 + d16	0 0 0 1 1 1 1 0 d<7:0> d<15:8>

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

CCF

<Complement Carry Flag>

Operation: CY ← inverted value of CY

Description: Inverts the contents of the carry flag.

Details:

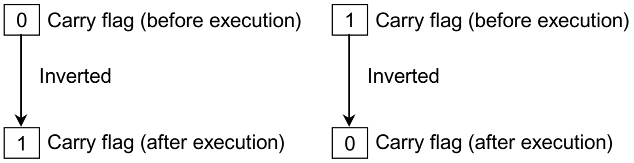
Mnemonic	Code
CCF	0 0 0 1 0 0 1 0

Flags:

S	Z	H	V	N	C
–	–	×	–	0	*

S = No change
Z = No change
H = An undefined value is set.
V = No change
N = Reset to 0.
C = Inverted value of itself is set.

Execution example: When the carry flag = 0, executing CCF sets the carry flag to 1; executing CCF again sets the carry flag to 0.


0 Carry flag (before execution)
Inverted
1 Carry flag (after execution)
1 Carry flag (before execution)
Inverted
0 Carry flag (after execution)

CHG num, dst

<Change>

Operation: dst<num> ← Inverted value of dst<num>

Description: Inverts the value of bit num of dst.

Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
○	○	×	CHG	#4, r	1 1 0 z 1 r 0 0 1 1 0 0 1 0 0 0 0 0 # 4
○	×	×	CHG	#3, (mem)	1 m 1 1 m m m m 1 1 0 0 0 #3

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: CHG 5, (100H)
When the contents of memory address 100 = 00100111B (binary), execution sets the contents to 00000111B (binary).

7	6	5	4	3	2	1	0	
0	0	1	0	0	1	1	1	Address 100 (before execution)

↓ Inverted

0	0	0	0	0	1	1	1	Address 100 (after execution)

CP src1, src2

<Compare>

Operation: src1 – src2

Description: Compares the contents of src1 with those of src2 and indicates the results in flag register F.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	○	CP	R, r	1 1 z z 1 r 1 1 1 1 0 R
○	○	×	CP	r, #3	1 1 0 z 1 r 1 1 0 1 1 #3
○	○	○	CP	r, #	1 1 z z 1 r 1 1 0 0 1 1 1 1 #<7:0> #<15:8> #<23:16> #<31:24>
○	○	○	CP	R, (mem)	1 m z z m m m m 1 1 1 1 0 R
○	○	○	CP	(mem), R	1 m z z m m m m 1 1 1 1 1 R
○	○	×	CP <W>	(mem), #	1 m 0 z m m m m 0 0 1 1 1 1 1 1 #<7:0> #<15:8>

Note: #3 in operands indicates from 0 to 7.

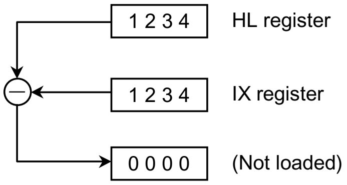
Flags:

S	Z	H	V	N	C
*	*	*	*	1	*

S = MSB value of the result is set.
Z = 1 is set if the result is 0, otherwise 0.
H = 1 is set if a borrow from bit 3 to bit 4 occurs as a result of the operation, otherwise 0. If the operand is 32 bits, an undefined value is set.
V = 1 is set if an overflow occurs as a result of the operation, otherwise 0.
N = 1 is set.
C = 1 is set if a borrow occurs from the MSB bit as a result of the operation, otherwise 0.

Execution example: CP HL,IX

When the HL register = 1234H and the IX register = 1234H, execution sets the Z and N flags to 1 and clears the S, H, V, and C flags to zero.



CPD src1, src2

<Compare Decrement>

Operation: src1 – src2, BC ← BC – 1

Description: Compares the contents of src1 with those of src2, then decrements the contents of the BC register by 1. src1 must be the A or WA register. src2 must be in post–decrement register indirect addressing mode.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	×	CPD	[A/WA, (R–)]	1 0 0 z 0 R 0 0 0 1 0 1 1 0

Note: Omitting operands in square brackets [ ] specifies A,(XHL–).

Flags:

S	Z	H	V	N	C
*	*	*	*	1	–

S = MSB value of the result of src1 – src2 is set.
Z = 1 is set if the result of src1 – src2 is 0, otherwise 0.
H = 1 is set if a borrow from bit 3 to bit 4 occurs as a result of src1 – src2, otherwise 0.
V = 0 is set if the BC register value is 0 after execution, otherwise 1.
N = 1 is set.
C = No change

Execution example: CPD A, (XIX–)

When the XIX register = 00123456H and the BC register = 0200H, execution compares the contents of the A register with those of memory address 123456H, then sets the XIX register to 00123455H, the BC register to 01FFH.

CPDR src1, src2

<Compare Decrement Repeat>

Operation: src1 – src2, BC ← BC – 1, Repeat until src1 = src2 or BC = 0

Description: Compares the contents of src1 with those of src2. Then decrements the contents of the BC register by 1. Repeats until src1 = src2 or BC = 0. src1 must be the A or WA register. src2 must be in post-decrement register indirect addressing mode.

Details:

Size			Mnemonic		Code							
Byte	Word	Long word										
○	○	×	CPDR	[A/WA, (R–)]	1	0	0	z	0	R		
					0	0	0	1	0	1	1	1

Note: Omitting operands in square brackets [ ] specifies A,(XHL–).

Flags:

S	Z	H	V	N	C
*	*	*	*	1	–

S = MSB value of the result of src1 – src2 is set.
Z = 1 is set if the result of src1 – src2 is 0, otherwise 0.
H = 1 is set if a borrow from bit 3 to bit 4 occurs as a result of src1 – src2, otherwise 0.
V = 0 is set if the BC register value is 0 after execution, otherwise 1.
N = 1 is set.
C = No change

Execution example: CPDR A,(XIX–)

Under the following conditions, execution reads the contents of memory addresses 123456H, 123455H, and 123454H. The instruction ends with condition BC = 0 and sets the XIX register to 00123453H and the BC register to 0000H.

Conditions: A register = 55H
XIX register = 00123456H
BC register = 0003H
Memory address 123456H = 11H
Memory address 123455H = 22H
Memory address 123454H = 33H

CPI src1, src2

<Compare Increment>

Operation: src1 – src2, BC ← BC – 1

Description: Compares the contents of src1 with those of src2, then decrements the contents of the BC register by 1. src1 must be the A or WA register. src2 must be in post-increment register indirect addressing mode.

Details:

Size			Mnemonic		Code							
Byte	Word	Long word										
○	○	×	CPI	[A/WA, (R+)]	1	0	0	z	0	R		
					0	0	0	1	0	1	0	0

Note: Omitting operands enclosed in square brackets [ ] specifies A,(XHL+).

Flags:

S	Z	H	V	N	C
*	*	*	*	1	–

S = MSB value of the result of src1 – src2 is set.
Z = 1 is set if the result of src1 – src2 is 0, otherwise 0.
H = 1 is set if a borrow from bit 3 to bit 4 occurs as a result of src1 – src2, otherwise 0.
V = 0 is set if the BC register value is 0 after execution, otherwise 1.
N = 1 is set.
C = No change

Execution example: CPI A, (XIX+)

When the XIX register = 00123456H and the BC register = 0200H, execution compares the contents of the A register with those of memory address 123456H, and sets the XIX register to 00123457H and the BC register to 01FFH.

CPIR src1, src2

<Compare Increment Repeat>

Operation: src1 − src2, BC ← BC − 1, repeat until src1 = src2 or BC = 0

Description: Compares the contents of src1 with those of src2. Then decrements the contents of the BC register by 1. Repeats until src1 = src2 or BC = 0. src1 must be the A or WA register. src2 must be in post-increment register indirect addressing mode.

Details:

Size			Mnemonic		Code							
Byte	Word	Long word										
○	○	×	CPIR	[A/WA, (R+)]	1	0	0	z	0	R		
					0	0	0	1	0	1	0	1

Note: Omitting operands in square brackets [ ] specifies A,(XHL+).

Flags:

S	Z	H	V	N	C
*	*	*	*	1	–

S = MSB value of the result of src1 − src2 is set.
Z = 1 is set if the result of src1 − src2 is 0, otherwise 0.
H = 1 is set if a borrow from bit 3 to bit 4 occurs as a result of src1 − src2, otherwise 0.
V = 0 is set if the BC register value is 0 after execution, otherwise 1.
N = 1 is set.
C = No change

Execution example: CPIR A, (XIX+)

Under the following conditions, execution reads memory addresses 123456H, 123457H, and 123458H. The instruction ends with condition src1 = src2, sets the XIX register to 00123459H and the BC register to 01FDH.

Conditions: A register = 33H
XIX register = 00123456
HBC register = 0200H
Memory address 123456H = 11H
Memory address 123457H = 22H
Memory address 123458H = 33H

CPL dst

<Complement>

Operation: dst ← Ones complement of dst

Description: Transfers the value of ones complement (inverted bit of 0/1) of dst to dst.

Details:

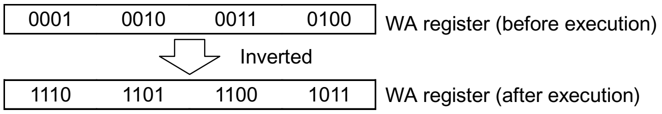
Size Byte	 Word	 Long word	Mnemonic		Code
○	○	×	CPL	r	1 1 0 z 1 r 0 0 0 0 0 1 1 0

Flags:

S	Z	H	V	N	C
–	–	1	–	1	–

S = No change
Z = No change
H = 1 is set.
V = No change
N = 1 is set.
C = No change

Execution example: CPL WA
When the WA register = 1234H, execution sets the WA register to EDCBH.



0001 0010 0011 0100 WA register (before execution)

Inverted

1110 1101 1100 1011 WA register (after execution)

DAA dst

<Decimal Adjust Accumulator>

Operation: dst ← decimal adjustment of dst

Description: Decimal adjusts the contents of dst depending on the states of the C, H, and N flags. Used to adjust the execution result of the add or subtract instruction as binary-coded decimal (BCD).

Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
○	×	×	DAA	r	1 1 0 0 1 r 0 0 0 1 0 0 0 0

Operation	N flag before DAA instruction execution	C flag before DAA instruction execution	Upper 4 bits of dst	H flag before DAA instruction execution	Lower 4 bits of dst	Added value	C flag after DAA instruction execution
	0	0	0 to 9	0	0 to 9	00	0
	0	0	0 to 8	0	A to F	06	0
ADD	0	0	0 to 9	1	0 to 3	06	0
	0	0	A to F	0	0 to 9	60	1
ADC	0	0	9 to F	0	A to F	66	1
	0	0	A to F	1	0 to 3	66	1
	0	1	0 to 2	0	0 to 9	60	1
	0	1	0 to 2	0	A to F	66	1
	0	1	0 to 3	1	0 to 3	66	1
SUB	1	0	0 to 9	0	0 to 9	00	0
SBC	1	0	0 to 8	1	6 to F	FA	0
NEG	1	1	7 to F	0	0 to 9	A0	1
	1	1	6 to F	1	6 to F	9A	1

Note: Decimal adjustment cannot be performed for the INC or DEC instruction. This is because the C flag does not change.

Flags:

S	Z	H	V	N	C
*	*	*	*	–	*

S = MSB value of the result is set.
Z = 1 is set if the result is 0, otherwise 0.
H = 1 is set if a carry from bit 3 to bit 4 occurs as a result of the operation, otherwise 0.
V = 1 is set if the parity (number of 1s) of the result is even, otherwise 0.
N = No change
C = 1 is set if a carry occurs from the MSB as a result of the operation or a carry was 1 before operation, otherwise 0.

Execution example:

```
ADD  A,B
DAA  A
```

When the A register = 59H and the B register = 13H,
execution sets the A register to 72H.

DEC num, dst

<Decrement>

Operation: dst ← dst – num

Description: Decrements dst by the contents of num and transfers the result to dst.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	○	DEC	#3, r	1 1 z z 1 | r 0 1 1 0 1 | #3
○	○	×	DEC<W>	#3, (mem)	1 m 0 z m m m m 0 1 1 0 1 | #3

Note: #3 in operands indicates from 1 to 8; object codes correspond from 1 to 7,0.

Flags:

S	Z	H	V	N	C
*	*	*	*	1	–

S = MSB value of the result is set.
Z = 1 is set if the result is 0, otherwise 0.
H = 1 is set if a borrow from bit 3 to bit 4 occurs as a result of the operation, otherwise 0.
V = 1 is set if an overflow occurs as a result of the operation, otherwise 0.
N = 1 is set.
C = No change

Note: With the DEC #3, r instruction, if the operand is a word or a long word, no flags change.

Execution example: DEC 4, HL
When the HL register = 5678H, execution sets the HL register to 5674H.

DECF

<Decrement Register File Pointer>

Operation: RFP<2:0> ← RFP<2:0> – 1

Description: Decrements the contents of register file pointer RFP <2:0> in the status register by 1. RFP2 is fixed to 0.

Details:

Mnemonic	Code
DECF	0 0 0 0 1 1 0 1

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: DECF

When the contents of RFP<2:0> = 2, execution sets the contents of RFP<2:0> to 1.

DI

<Disable Interrupt>

Operation: IFF<2:0> ← 7

Description: Sets the contents of the interrupt enable flag (IFF) <2:0> in status register to 7. After execution, only non-maskable interrupts (interrupt level 7) can be received.

Details:

Mnemonic	Code
DI	0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

DIV dst, src

<Divide>

Operation: dst<lower half> ← dst ÷ src, dst<upper half> ← remainder (unsigned)

Description: Divides unsigned the contents of dst by those of src and transfers the quotient to the lower half of dst, the remainder to the upper half of dst.

Details:

	Size		Mnemonic		Code							
Byte	Word	Long word										
○	○	×	DIV	RR, r	1	1	0	z	1	r		
					0	1	0	1	0	R		
○	○	×	DIV	rr, #	1	1	0	z	1	r		
					0	0	0	0	1	0	1	0
					#<7:0>							
					#<15:8>							
○	○	×	DIV	RR, (mem)	1	m	0	z	m	m	m	m
					0	1	0	1	0	R		

Note 1: For RR, see the following page.

Note 2: When the operation is in bytes, dst (lower byte) ← dst (word) ÷ src (byte), dst (upper byte) ← remainder.
When the operation is in words, dst (lower word) ← dst (long word) ÷ src (word), dst (upper word) ← remainder. Match coding of the operand dst with the size of the dividend.

Flags:

S	Z	H	V	N	C
–	–	–	V	–	–

S = No change
Z = No change
H = No change
V = 1 is set when divided by 0 or the quotient exceeds the numerals which can be expressed in bits of dst for load; otherwise, 0 is set.
N = No change
C = No change

Execution example: DIV XIX,IY

When the XIX register = 12345678H and the IY register = 89ABH, execution results in a quotient of 21DAH and a remainder of 0FDAH, and sets the XIX register to 0FDA21DAH.

Note 3: RR of the DIV RR,r and DIV RR,(mem) instruction is as listed below.

Operation size in bytes
(8 bits ← 16 bits ÷ 8 bits)

RR	Code R
WA	001
BC	011
DE	101
HL	111
IX	Specification not possible!
IY	
IZ	
SP	

Operation size in words
(16 bits ← 32 bits ÷ 16 bits)

RR	Code R
XWA	000
XBC	001
XDE	010
XHL	011
XIX	100
XIY	101
XIZ	110
XSP	111

*1 When the CPU is in minimum mode, XWA, XBC, XDE, and XHL cannot be used.

rr of the DIV rr,# instruction is as listed below.

Operation size in bytes
(8 bits ← 16 bits ÷ 8 bits)

rr	Code r
WA	001
BC	011
DE	101
HL	111
IX	C7H : F0H
IY	C7H : F4H
IZ	C7H : F8H
SP	C7H : FCH
	1st byte 2nd byte

*2 Any other word registers can be specified in thesame extension coding as IX to SP.

Operation size in words
(16 bits ← 32 bits ÷ 16 bits)

rr	Code r
XWA	000
XBC	001
XDE	010
XHL	011
XIX	100
XIY	101
XIZ	110
XSP	111

*3 When the CPU is in minimum mode, XWA, XBC, XDE, and XHL cannot be used.

*4 Any other long word registers can be specified in the extension coding.

DIVS dst, src

<Divide Signed>

Operation: dst<lower half> ← dst ÷ src,dst<upper half> ← remainder (signed)

Description: Divides signed the contents of dst by those of src and transfers the quotient to the lower half of dst, the remainder to the upper half of dst.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	×	DIVS	RR, r	1 1 0 z 1 r 0 1 0 1 1 R
○	○	×	DIVS	rr, #	1 1 0 z 1 r 0 0 0 0 1 0 1 1 #<7:0> #<15:8>
○	○	×	DIVS	RR, (mem)	1 m 0 z m m m m 0 1 0 1 1 R

Note 1: For RR, see the following page.

Note 2: When the operation is in bytes, dst (lower byte) ← dst (word) ÷ src (byte), dst (upper byte) ← remainder.
When the operation is in words, dst (lower word) ← dst (long word) ÷ src (word), dst (upper word) ← remainder.
Match coding of the operand dst with the size of the dividend. The sign of the remainder is the same as that of the dividend.

Flags:

S	Z	H	V	N	C
–	–	–	*	–	–

S = No change
Z = No change
H = No change
V = 1 is set when divided by 0, or the quotient exceeds the value which can be expressed in bits of the dst used for loading, otherwise 0.
N = No change
C = No change

Execution example: DIVS XIX,IY
When the XIX register = 12345678H and the IY register = 89ABH, execution results in the quotient as 16EEH and the remainder as D89EH, and sets the XIX register to 16EED89EH.

Note 3: RR of the DIVS RR,r and DIVS RR, (mem) instruction is as listed below.

Operation size in bytes
(8 bits ← 16 bits ÷ 8 bits)

RR	Code R
WA	001
BC	011
DE	101
HL	111
IX	Specification not possible!
IY	
IZ	
SP	

Operation size in words
(16 bits ← 32 bits ÷ 16 bits)

RR	Code R
XWA	000
XBC	001
XDE	010
XHL	011
XIX	100
XIY	101
XIZ	110
XSP	111

*1 When the CPU is in minimum mode, XWA, XBC, XDE, or XHL cannot be used.

rr of the DIVS rr,# instruction is as listed below.

Operation size in bytes
(8 bits ← 16 bits ÷ 8 bits)

rr	Code r
WA	001
BC	011
DE	101
HL	111
IX	C7H : F0H
IY	C7H : F4H
IZ	C7H : F8H
SP	C7H : FCH
	1st byte 2nd byte

*2 Any other word registers can be specified in the same extension coding as those for IX to SP.

Operation size in words
(16 bits ← 32 bits ÷ 16 bits)

rr	Code r
XWA	000
XBC	001
XDE	010
XHL	011
XIX	100
XIY	101
XIZ	110
XSP	111

*3 When the CPU is in minimum mode, XWA, XBC, XDE, or XHL cannot be used.

*4 Any other long word registers can be specified in the extension coding.

DJNZ dst1, dst2

<Decrement and Jump if Non Zero>

Operation: dst1 ← dst1 − 1. if dst1 ≠ 0, then PC ← dst2.

Description: Decrements the contents of dst1 by 1. Makes a relative jump to the program address specified by dst2 if the result is other than 0.

Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
○	○	×	DJNZ	[r,]$ + 3/4 + d8	1 1 0 z 1 r 0 0 0 1 1 1 0 0 d<7:0>

(Note) $ + 4 + d8 (r is specified using extension codes.)
$ + 3 + d8 (otherwise)

Note: Omitting r of the operand in square brackets [ ] is regarded as specifying the B register.

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example:

```
LOOP:  ADD  A, A
       DJNZ  W, LOOP
```

When the A register = 12H and the W register = 03H, execution loops three times and sets the A register to 24H → 48 → 90H and the W register to 02H → 01H → 00H.

EI num

<Enable Interrupt>

Operation: IFF <2:0> ← num

Description: Sets the contents of the IFF<2:0> in the status register to num. After execution, the CPU interrupt receive level becomes num.

Details:

Mnemonic		Code
EI	[#3]	0 0 0 0 0 1 1 0 0 0 0 0 0 #3

Note: A value from 0 to 7 can be specified as the operand value. If the operand is omitted, the default value is 0 (EI 0).

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

EX dst, src

<Exchange>

Operation: dst ↔ src

Description: Exchanges the contents of dst and src.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	×	×	EX	F, F’	0 0 0 1 0 1 1 0
○	○	×	EX	R, r	1 1 z z 1 r 1 0 1 1 1 R
○	○	×	EX	(mem), r	1 m z z m m m m 0 0 1 1 0 R

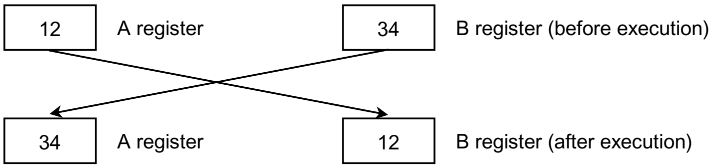
Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change
Note: Executing EX F,F’ changes all flags.

Execution example: EX A,B

When the A register = 12H and the B register = 34H, execution sets the A register to 34H and the B register to 12H.



EXTS dst
<Extend Sign>

Operation: dst <upper half> ← signed bit of dst <lower half>

Description: Transfers (copies) the signed bit (bit 7 when the operand size is a word, bit 15 when a long word) of the lower half of dst to all bits of the upper half of dst.

Details:

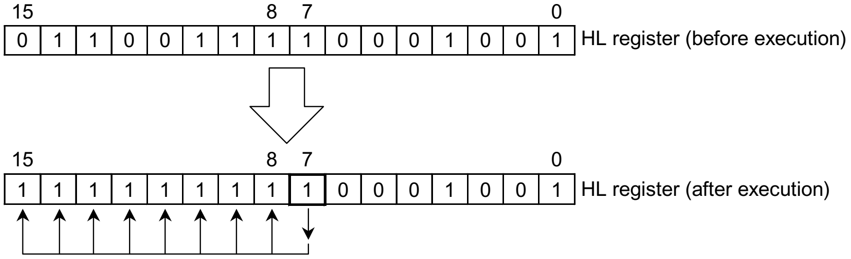
Size Byte	 Word	 Long word	Mnemonic		Code
×	○	○	EXTS	r	1 1 z z 1 r 0 0 0 1 0 0 1 1

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: EXTS HL
When the HL register = 6789H, execution sets the HL register to FF89H.



EXTZ dst

<Extend Zero>

Operation: dst<upper half> ← 0

Description: Clears the upper half of dst to zero. Used for making the operand sizes the same when they are different.

Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
×	○	○	EXTZ	r	1 1 z z 1 r
					0 0 0 1 0 0 1 0

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: EXTZ HL
When the HL register = 6789H, execution sets the HL register to 0089H.

EXTZ XIX
When the XIX register = 12345678H, execution sets the XIX register to 00005678H.

HALT

<Halt CPU>

Operation: CPU halt

Description: Halts the instruction execution. To resume, an interrupt must de received.

Details:

Mnemonic	Code
HALT	0 0 0 0 0 1 0 1

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

INC num, dst

<Increment>

Operation: dst ← dst + num

Description: Adds the contents of dst and num and transfers the result to dst.

Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
○	○	○	INC	#3, r	1 1 z z 1 r
					0 1 1 0 0 #3
○	○	×	INC<W>	#3, (mem)	1 m 0 z m m m m
					0 1 1 0 0 #3

Note: #3 in operands indicates from 1 to 8 and object codes correspond from 1 to 7,0.

Flags:

S	Z	H	V	N	C
*	*	*	*	0	–

S = MSB value of the result is set.
Z = 1 is set if the result is 0, otherwise 0.
H = 1 is set if a carry occurs from bit 3 to bit 4 as a result of the operation, otherwise 0.
V = 1 is set if an overflow occurs as a result of the operation, otherwise 0.
N = Cleared to zero.
C = No change

Note: With the INC #3,r instruction, if the operand is a word or a long word, no flags change.

Execution example: INC 5,WA
When the WA register = 1234H, execution sets the WA register to 1239H.

INCF

<Increment Register File Pointer>

Operation: RFP<2:0> ← RFP<2:0> + 1

Description: Increments the contents of RFP<2:0> in the status register by 1. RFP2 is fixed to 0.

Details:

Mnemonic	Code
INCF	0 0 0 0 1 1 0 0

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: INCF
When the contents of RFP<2:0> = 2, execution sets the contents of RFP<2:0> to 3.

JP condition, dst

<Jump>

Operation: If cc is true, then PC ← dst.

Description: If the operand condition is true, jumps to the program address specified by dst.

Details:

Size	Mnemonic		Code
	JP	#16	0 0 0 1 1 0 1 0 #<7:0> #<15:8>
	JP	#24	0 0 0 1 1 0 1 1 #<7:0> #<15:8> #<23:16>
	JP	[cc,] mem	1 m 1 1 m m m m 1 1 0 1 c c

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: When JP 2000H is executed, jumps unconditionally to address 2000H. When the XIX register = 00123456H, and carry flag's value is 1, JP 2000H jumps to address 123458H by the execution of JP C and XIX+2.

JR condition, dst

<Jump Relative>

Operation: If cc is true, then PC ← dst.

Description: If the operand condition is true, makes a relative jump to the program address specified by dst.

Details:

Mnemonic		Code
JR	[cc,] $ + 2 + d8	0 1 1 0 c c d<7:0>
JRL	[cc,] $ + 3 + d16	0 1 1 1 c c #<7:0> #<15:8>

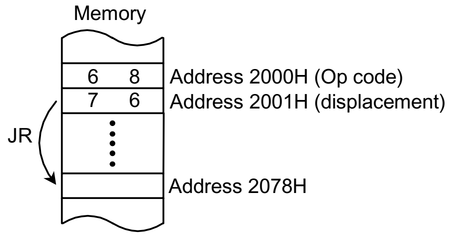
Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: JR 2078H

When this instruction is executed at memory address 2000H, execution relative jumps unconditionally to address 2078H. The object code of the instruction is 68H : 76H.



LD dst, src

<Load>

Operation: dst ← src

Description: Loads the contents of src to dst.

Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
○	○	○	LD	R, r	1 1 z z 1 r 1 0 0 0 1 R
○	○	○	LD	r, R	1 1 z z 1 r 1 0 0 1 1 R
○	○	○	LD	r, #3	1 1 z z 1 r 1 0 1 0 1 #3
○	○	○	LD	R, #	0 z z z 0 R #<7:0> #<15:8> #<23:16> #<31:24>
○	○	○	LD	r, #	1 1 z z 1 r 0 0 0 0 0 0 1 1 #<7:0> #<15:8> #<23:16> #<31:24>
○	○	○	LD	R, (mem)	1 m z z m m m m 0 0 1 0 0 R
○	○	○	LD	(mem), R	1 m 1 1 m m m m 0 1 z z 0 R
○	○	×	LD<W>	(#8), #	0 0 0 0 1 0 z 0 #8 #<7:0> #<15:8>

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	×	LD<W>	(mem), #	1 m 1 1 m m m m 0 0 0 0 0 0 z 0 #<7:0> #<15:8>
○	○	×	LD<W>	(#16), (mem)	1 m 0 z m m m m 0 0 0 1 1 0 0 1 #16<7:0> #16<15:8>
○	○	×	LD<W>	(mem), (#16)	1 m 1 1 m m m m 0 0 0 1 0 1 z 0 #16<7:0> #16<15:8>

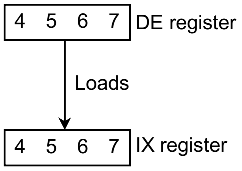
Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: LD IX, DE

When the DE register = 4567H, execution sets the IX register to 4567H.



LDA dst, src

<Load Address>

Operation: dst ← src effective address value

Description: Loads the src effective address value to dst.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
×	○	○	LDA	R, mem	1 m 1 1 m m m m 0 0 1 s 0 R

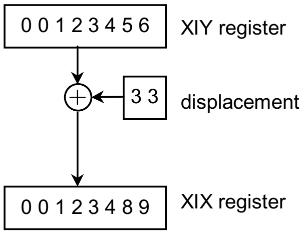
Note: This instruction operates much like the ADD instruction; the difference is that dst is specified independently from src. Mainly used for handling the pointer with the C compiler.

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: LDA XIX, XIY + 33H
When the XIY register = 00123456H, execution sets the XIX register to 00123489H.



LDAR dst, src

<Load Address Relative>

Operation: dst ← src relative address value

Description: Loads the relative address value specified in src to dst.

Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
×	○	○	LDAR	R, $ + 4 + d16	1 1 1 1 0 0 1 1
					0 0 0 1 0 0 1 1
					d<7:0>
					d<15:8>
					0 0 1 s 0 R

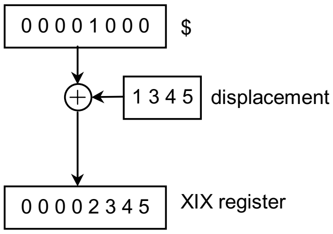
Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: LDAR XIX, $ + 1345H

When this instruction is executed at memory address 1000H, execution sets the XIX register to 00002345H. $ indicates the start address of the instruction. The instruction's object codes are: F3H:13H:41H:13H:34H.



LDC dst, src

<Load Control Register>

Operation: dst ← src

Description: Loads the contents of src to dst.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	○	LDC	cr, r	1 1 z z 1 r 0 0 1 0 1 1 1 0 cr
○	○	○	LDC	r, cr	1 1 z z 1 r 0 0 1 0 1 1 1 1 cr

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: LDC DMAC0, WA
When the WA register = 1234H, execution sets control register DMAC0 to 1234H.

LDCF num, src

<Load Carry Flag>

Operation: CY ← src<num>

Description: Loads the contents of bit num of src to the carry flag.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	×	LDCF	#4, r	1 1 0 z 1 r 0 0 1 0 0 0 1 1 0 0 0 0 # 4
○	○	×	LDCF	A, r	1 1 0 z 1 r 0 0 1 0 1 0 1 1
○	×	×	LDCF	#3, (mem)	1 m 1 1 m m m m 1 0 0 1 1 #3
○	×	×	LDCF	A, (mem)	1 m 1 1 m m m m 0 0 1 0 1 0 1 1

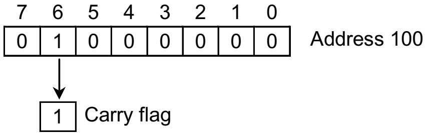
Notes: When bit num is specified by the A register, the value of the lower 4 bits of the A register is used as bit num. When the operand is a byte and the value of the lower 4 bits of bit num is from 8 to 15, the value of the carry flag is undefined.

Flags:

S	Z	H	V	N	C
–	–	–	–	–	*

S = No change
Z = No change
H = No change
V = No change
N = No change
C = Contents of bit num of src is set.

Execution example: LDCF 6, (100H)
When the contents of memoryad address 100 = 01000000B (binary), execution sets the carry flag to 1.



LDD dst, src

<Load Decrement>

Operation: dst ← src, BC ← BC − 1

Description: Loads the contents of src to dst, then decrements the contents of the BC register by 1. src and dst must be in post-decrement register indirect addressing mode.

Details:

	Size		Mnemonic	Code
Byte	Word	Long word		
○	○	×	LDD<W>[(XDE–), (XHL–)]	1 0 0 z 0 0 1 1 0 0 0 1 0 0 1 0
			LDD<W> (XIX–), (XIY–)	1 0 0 z 0 1 0 1 0 0 0 1 0 0 1 0

*Coding in square brackets [ ] can be omitted.

Flags:

S	Z	H	V	N	C
–	–	0	*	0	–

S = No change
Z = No change
H = Cleared to 0.
V = 0 is set if the BC register value is 0 after execution, otherwise 1.
N = Cleared to zero.
C = No change

Execution example: LDD (XIX–), (XIY–)
When the XIX register = 00123456H, the XIY register = 00335577H, and the BC register = 0700H, execution loads the contents at address 335577 to address 123456H and sets the XIX register to 123455H, the XIY register to 00335576H, and the BC register to 06FFH.

LDDR dst, src

<Load Decrement Repeat>

Operation: dst ← src, BC ← BC − 1, Repeat until BC = 0

Description: Loads the contents of src to dst, then decrements the contents of the BC register by 1. If the result is other than 0, the operation is repeated. src and dst must be in post-decrement register indirect addressing mode.

Details:

Size			Mnemonic	Code
Byte	Word	Long word		
○	○	×	LDDR<W>[(XDE−), (XHL−)]	1 0 0 z 0 0 1 1 0 0 0 1 0 0 1 1
○	○	×	LDDR<W> (XIX−), (XIY−)	1 0 0 z 0 1 0 1 0 0 0 1 0 0 1 1

* Coding in square brackets [ ] can be omitted.

Flags:

S	Z	H	V	N	C
−	−	0	0	0	−

S = No change
Z = No change
H = Cleared to zero.
V = Cleared to zero.
N = Cleared to zero.
C = No change

Execution example: LDDR (XIX−), (XIY−)

When the XIX register = 00123456H, the XIY register = 00335577H, and the BC register = 0003H, the results of the execution are as follows:

Loads the contents of address 335577H to 123456H.
Loads the contents of address 335576H to 123455H.
Loads the contents of address 335575H to 123454H.
Sets the XIX register to 00123453H.
Sets the XIY register to 00335574H.
Sets the BC register to 0000H.

LDF num

<Load Register File Pointer>

Operation: RFP<2:0> ← num

Description: Loads the num value to the register file pointer RFP<2:0> in status register. RFP2 is fixed to 0.

Details:

Mnemonic		Code
LDF	#3	0 0 0 1 0 1 1 1
		0 0 0 0 0 #3

Note: In minimum mode, the operand value can be specified from 0 to 7; in maximum mode, from 0 to 3.

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

LDI dst, src

<Load Increment>

Operation: dst ← src, BC ← BC − 1

Description: Loads the contents of src to dst, then decrements the contents of the BC register by 1. src and dst must be in post-increment register indirect addressing mode.

Details:

	Size		Mnemonic	Code							
Byte	Word	Long word									
○	○	×	LDI<W>[(XDE+), (XHL+)]	1	0	0	z	0	0	1	1
				0	0	0	1	0	0	0	0
○	○	×	LDI<W> (XIX+), (XIY+)	1	0	0	z	0	1	0	1
				0	0	0	1	0	0	0	0

Note: Coding in square brackets [ ] can be omitted.

Flags:

S	Z	H	V	N	C
–	–	0	*	0	–

S = No change
Z = No change
H = Cleared to zero.
V = 0 is set when the BC register value is 0 after execution, otherwise 1.
N = Cleared to zero.
C = No change

Execution example: LDI (XIX+), (XIY+)

When the XIX register = 00123456H, the XIY register = 00335577H, and the BC register = 0700H, execution loads the contents of address 335577H to 123456H and sets the XIX register to 00123457H, the XIY register to 00335578H, and the BC register to 06FFH.

LDIR dst, src

<Load Increment Repeat>

Operation: dst ← src, BC ← BC − 1, Repeat until BC = 0

Description: Loads the contents of src to dst, then decrements the contents of the BC register by 1. If the result is other than 0, the operation is repeated. src and dst must be in post-increment register indirect addressing mode.

Details:

Size			Mnemonic	Code							
Byte	Word	Long word									
○	○	×	LDIR<W>[(XDE+), (XHL+)]	1	0	0	z	0	0	1	1
				0	0	0	1	0	0	0	1
○	○	×	LDIR<W> (XIX+), (XIY+)	1	0	0	z	0	1	0	1
				0	0	0	1	0	0	0	1

Note: Coding in square brackets [ ] can be omitted.

Note: Interrupt requests are sampled every time 1 item of data is loaded.

Flags:

S	Z	H	V	N	C
–	–	0	0	0	–

S = No change
Z = No change
H = Cleared to zero.
V = Cleared to zero.
N = Cleared to zero.
C = No change

Execution example: LDIR (XIX+), (XIY+)

When the XIX register = 00123456H, the XIY register = 00335577H, and the BC register = 0003H, execution results as follows:

Loads the contents of address 335577H to 123456H.
Loads the contents of address 335578H to 123457H.
Loads the contents of address 335579H to 123458H.
Sets the XIX register to 00123459H.
Sets the XIY register to 0033557AH.
Sets the BC register to 0000H.

LDX dst, src

<Load eXtract>

Operation: dst ← src

Description: Loads the contents of src to dst. The effective code is assigned to this instruction every other byte. Used to fetch the code from 8-bit data bus memory in 16-bit data bus mode.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	×	×	LDX	(#8), #	1 1 1 1 0 1 1 1
					0 0 0 0 0 0 0 0
					#8
					0 0 0 0 0 0 0 0
					#
					0 0 0 0 0 0 0 0

Note: Even if the second, fourth, or sixth instruction code value is not 00H, the instruction operates correctly.

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

This instruction is used when the CPU fetches a program after reset in cases where the bus width set in the 900/L1 is 16 bits wide and that of external program ROM is 8 bits wide.

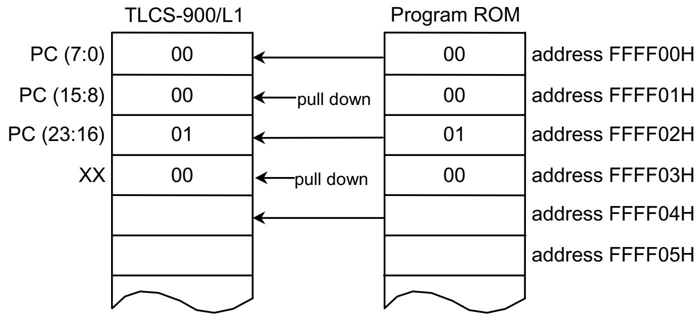
The table below shows usage conditions.

Product Name	AM0 Pin	AM1 Pin	Program ROM Bus Width	Other Memory Bus Width	LDX Instruction
TMP91C815	0	0	8-bit	8-bit	Not used
	1	0	8-bit	8/16-bit	Used
			16-bit	8/16-bit	Not used
	1	1	Internal ROM	–	–

Execution example: Explanation here is given using the TMP91C815 as an example where while AM0, 1 = 1, 0 and all memory but program ROM are 16 bits wide, the instruction is executed from 8-bit wide program ROM. After reset, the reset vector is read in 16-bit data bus mode. Therefore, when starting the program from external memory with 8-bit data bus, the PC (15:8) value for the reset vector must be entered by connecting pull-up/down resistors to the upper-byte data bus D8 to D15 pins.

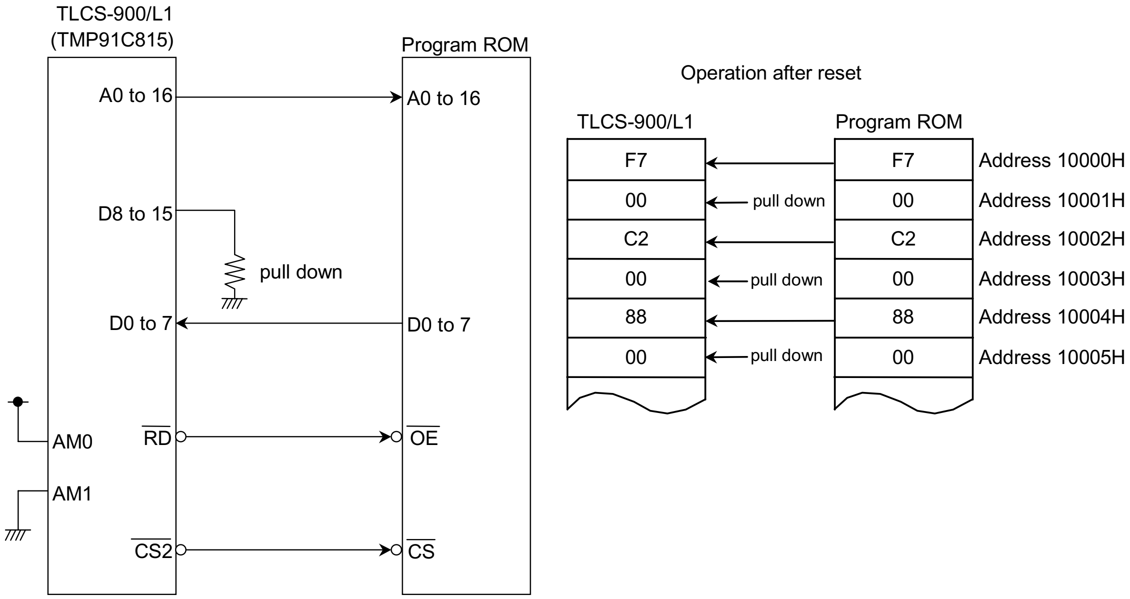
For example, if the reset vector is located at address 010000H, place 010000H in program ROM address FFFF00H and pull the D8 to D15 pins low. This allows the value 00H to be entered for the PC (15:8). Then place the LDX instruction in program ROM address 010000H.

Operation immediately after reset



LDX (0C2H), 88H

When the above instruction is executed, the CPU writes data 88H to the control register at address 0C2H of the internal programmable chip select/wait controller. As a result, the CS2 space is placed in 8-bit data bus 2WAIT mode, so that the program is fetched and executed via an 8-bit bus beginning with the next instruction.



Note: The pull-up/down added to the D8 to D15 pins to enter the reset vector PC (15:8) results in colliding with data outputs from the D8 to D15 pins, causing the current consumption to increase. Therefore, if this presents a problem, the pull-up/down must be disconnected after the above processing is finished.

LINK dst, num

<Link>

Operation: (–XSP) ← dst, dst ← XSP, XSP ← XSP + num

Description: Saves the contents of dst to the stack area. Loads the contents of stack pointer XSP to dst. Adds the contents of XSP to those of num (signed) and loads the result to XSP. Used for obtaining a local variable area in the stack area for -num bytes.

Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
×	×	○	LINK	r,d16	1 1 1 0 1 r
					0 0 0 0 1 1 0 0
					d<7:0>
					d<15:8>

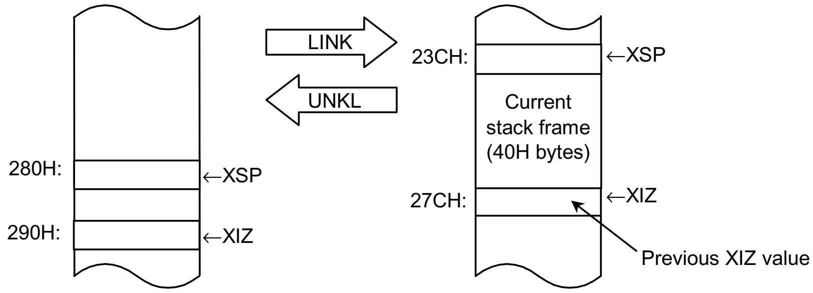
Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: LINK XIZ, −40H

When stack pointer XSP = 280H and the XIZ register = 290H, execution writes 00000290H (long data) at memory address 27CH and sets the XIZ register to 27CH and the stack pointer to XSP 23CH.



MDEC1 num, dst

<Modulo Decrement 1>

Operation: if (dst mod num) = 0 then dst ← dst + (num – 1) else dst ← dst – 1.

Description: When the modulo num of dst is 0, increments dst by num –1 .
Otherwise, decrements dst by 1. Used to operate pointers for cyclic memory table.

Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
×	○	×	MDEC1	#, r	1 1 0 1 1 r 0 0 1 1 1 1 0 0 #<7:0> – 1 #<15:8>

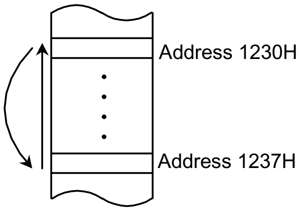
Note: The operand # must be 2 to the nth power. (n = 1 to 15)

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: Decrements the IX register by cycling from 1230H to 1237H. MDEC1 8, IX When the IX register = 1231H, execution sets the IX register to 1230H. Further execution increments the IX register by 8 – 1 and sets the IX register to 1237H, since the IX register modulo 8 = 0.



MDEC2 num, dst

<Modulo Decrement 2>

Operation: if (dst mod num) = 0 then dst ← dst + (num – 2) else dst ← dst – 2.

Description: When the modulo num of dst is 0, increments dst by num –2.
Otherwise, decrements dst by 2. Used to operate pointers for cyclic memory table.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
×	○	×	MDEC2	#, r	1 1 0 1 1 | r
					0 0 1 1 1 1 0 1
					#<7:0> – 2
					#<15:8>

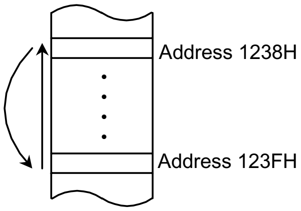
Note: The operand # must be 2 to the nth power. (n = 2 to 15)

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: Decrements the IX register by cycling from 1238H to 123FH. MDEC2 8,IX
When the IX register = 123AH, execution sets the IX register to 1238H. Further execution increments the IX register by 8 – 2 and sets the IX register to 123EH, since the IX register modulo 8 = 0.



MDEC4 num, dst

<Modulo Decrement 4>

Operation: if (dst mod num) = 0 then dst ← dst + (num − 4) else dst ← dst − 4.

Description: When the modulo num of dst is 0, increments dst by num −4. Otherwise, decrements dst by 4. Used to operate pointers for cyclic memory table.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
×	○	×	MDEC4	#, r	1 1 0 1 1 | r
					0 0 1 1 1 1 1 0
					#<7:0> – 4
					#<15:8>

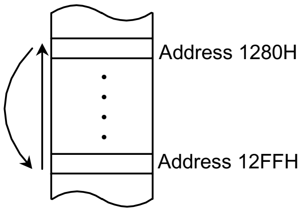
Note: The operand # must be 2 to the nth power. (n = 3 to 15)

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: Decrements the IX register by cycling from 1280H to 12FFH. MDEC4 80H,IX
When the IX register = 1284H, execution sets the IX register to 1280H. Further execution increments the IX register by 80H − 4 and sets the IX register to 12FCH, since the IX register modulo 80H = 0.



MINC1 num, dst

<Modulo Increment 1>

Operation: if (dst mod num) = (num − 1) then dst ← dst − (num − 1) else dst ← dst + 1.

Description: When the modulo num of dst is num − 1, decrements dst by num − 1. Otherwise, increments dst by 1. Used to operate pointers for cyclic memory table .

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
×	○	×	MINC1	#, r	1 1 0 1 1 r 0 0 1 1 1 0 0 0 #<7:0> – 1 #<15:8>

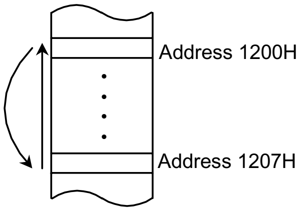
Note: The operand # must be 2 to the nth power. (n = 1 to 15)

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: Increments the IX register by cycling from 1200H to 1207H. MINC1 8, IX When the IX register = 1206H, execution sets the IX register to 1207H. Further execution decrements the IX register by 8 − 1 and sets the IX register to 1200H, since the IX register modulo 8 = 8 − 1.



MINC2 num, dst

<Modulo Increment 2>

Operation: if (dst mod num) = (num – 2) then dst ← dst – (num – 2) else dst ← dst + 2.

Description: When the modulo num of dst is num – 2, decrements dst by num – 2. Otherwise, increments dst by 2. Used to operate pointers for cyclic memory table.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
×	○	×	MINC2	#, r	1 1 0 1 1 | r 0 0 1 1 1 0 0 1 #<7:0> – 2 #<15:8>

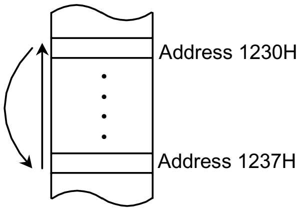
Note: The operand # must be 2 to the nth power. (n = 2 to 15)

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: Increments the IX register by cycling from 1230H to 1237H. MINC2 8,IX When the IX register = 1234H, execution sets the IX register to 1236H. Further execution decrements the IX register by 8 – 2 and sets the IX Register to 1230H, since the IX register modulo 8 = 8 – 2.



MINC4 num, dst

<Modulo Increment 4>

Operation: if (dst mod num) = (num − 4) then dst ← dst − (num − 4) else dst ← dst + 4.

Description: When the modulo num of dst is num − 4, decrements dst by num − 4. Otherwise, increments dst by 4. Used to operate pointers for cyclic memory table.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
×	○	×	MINC4	#, r	1 1 0 1 1 | r
					0 0 1 1 1 0 1 0
					#<7:0> – 4
					#<15:8>

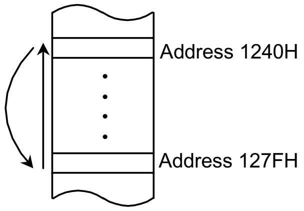
Note: The operand # must be 2 to the nth power. (n = 3 to 15)

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change

Execution example: Increments the IX register by cycling from 1240H to 127FH. MINC4 40H,IX
When the IX register = 1278H, execution sets the IX register to 127CH. Further execution decrements the IX register by 40H − 4 and sets the IX register to 1240H, since the IX register modulo 40H = 40H − 4.



MIRR dst

<Mirror>

Operation: dst<MSB:LSB> ← dst<LSB:MSB>

Description: Mirror-exchanges the contents of dst using the bit pattern image.

Details:

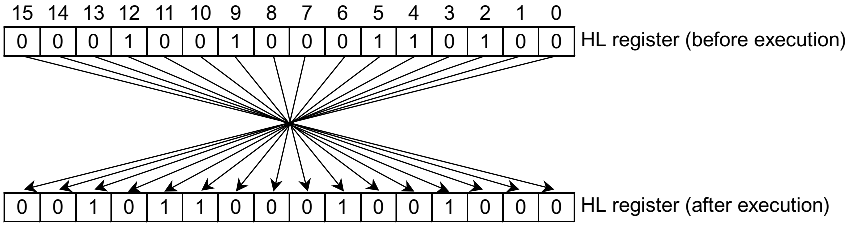
	Size		Mnemonic		Code
Byte	Word	Long word			
×	○	×	MIRR	r	1 1 0 1 1 r 0 0 0 1 0 1 1 0

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change

Execution example: MIRR HL
When the HL register = 0001 0010 0011 0100B (binary), execution sets the HL register to 0010 1100 0100 1000B (binary).



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0	HL register (before execution)
0	0	1	0	1	1	0	0	0	1	0	0	1	0	0	0	HL register (after execution)

MUL dst, src

<Multiply>

Operation: dst ← dst<lower half> × src (unsigned)

Description: Multiplies unsigned the contents of lower half of dst by those of src and loads the result to dst.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	×	MUL	RR, r	1 1 0 z 1 r 0 1 0 0 0 R
○	○	×	MUL	rr, #	1 1 0 z 1 r 0 0 0 0 1 0 0 0 #<7:0> #<15:8>
○	○	×	MUL	RR, (mem)	1 m 0 z m m m m 0 1 0 0 0 R

Note: When the operation is in bytes, dst (word) ← dst (byte) × src (byte).
When the operation is in words, dst (long word) ← dst (word) × src (word).
Match coding of the operand dst with the size of the result.

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change

Execution example: MUL XIX, IY
When the IX register = 1234H and the IY register = 89ABH, execution multiplies unsigned the contents of the IX register by those of the IY register and sets the XIX register to 09C9FCBCH.

Note: RR for the MUL RR,r and MUL RR, (mem) instructions is as listed below:

Operation size in bytes
(16 bits ← 8 bits × 8 bits)

RR	Code R
WA	001
BC	011
DE	101
HL	111
IX	Specification not possible!
IY	
IZ	
SP	

Operation size in words
(32 bits ← 16 bits × 16 bits)

RR	Code R
XWA	000
XBC	001
XDE	010
XHL	011
XIX	100
XIY	101
XIZ	110
XSP	111

*1 When the CPU is in minimum mode, XWA, XBC, XDE, or XHL cannot be used.

rr of the MUL rr,# instruction is as listed below.

Operation size in bytes
(16 bits ← 8 bits × 8 bits)

rr	Code r
WA	001
BC	011
DE	101
HL	111
IX	C7H : F0H
IY	C7H : F4H
IZ	C7H : F8H
SP	C7H : FCH
	1st byte 2nd byte

*2 Any other word registers can be specified in the same extension coding as those for IX to SP.

Operation size in words
(32 bits ← 16 bits × 16 bits)

rr	Code r
XWA	000
XBC	001
XDE	010
XHL	011
XIX	100
XIY	101
XIZ	110
XSP	111

*3 When the CPU is in minimum mode, XWA, XBC, XDE, or XHL cannot be used.

*4 Any other long word registers can be specified in the extension coding.

MULA dst

<Multiply and Add>

Operation: dst ← dst + (XDE) × (XHL), XHL ← XHL – 2

Description: Multiplies signed the memory data (16 bits) specified by the XDE register by the memory data (16 bits) specified by the XHL register . Adds the result (32 bits) to the contents of dst (32 bits) and loads the sum to dst (32 bits). Then, decrements the contents of the XHL register by 2.

Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
×	○	×	MULA	rr	1 1 0 1 1 | r 0 0 0 1 1 0 0 1

Note: Match coding of the operand dst with the operation size (long word).

Flags:

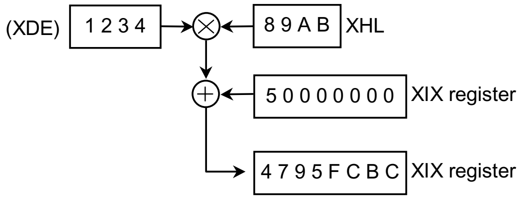
S	Z	H	V	N	C
*	*	–	*	–	–

S = MSB value of the result is set.
Z = 1 is set when the result is 0, otherwise 0.
H = No change.
V = 1 is set when an overflow occurs as a result, otherwise 0.
N = No change.
C = No change.

Execution example: MULA XIX

Under the following conditions, execution sets the XIX register to 4795FCBCH and the XHL register to 1FEH.

Conditions: XIX register = 50000000H
XDE register = 100H
XHL register = 200H
Memory data (word) at address 100H = 1234H
Memory data (word) at address 200H = 89ABH



MULS dst, src

<Multiply Signed>

Operation: dst ← dst<lower half> × src (signed)

Description: Multiplies signed the contents of the lower half of dst by those of src and loads the result to dst.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	×	MULS	RR, r	1 1 0 z 1 r 0 1 0 0 1 R
○	○	×	MULS	rr, #	1 1 0 z 1 r 0 0 0 0 1 0 0 1 #<7:0> #<15:8>
○	○	×	MULS	RR, (mem)	1 m 0 z m m m m 0 1 0 0 1 R

Note: When the operation is in bytes, dst(word) ← dst (byte) × src (byte).
When the operation is in words, dst (long word) ← dst (word) × src (word).
Match coding of the operand dst with the size of the result.

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change

Execution example: MULS XIX, IY

When the IX register = 1234H and the IY register = 89ABH, execution multiplies signed the contents of the IX register by those of the IY register and sets the XIX register to F795FCBCH.

Note: RR for the MULS RR,r and MULS RR, (mem) instructions is as listed below:

Operation size in bytes
(16 bits ← 8 bits × 8 bits)

RR	Code R
WA	001
BC	011
DE	101
HL	111
IX	Specification not possible!
IY	
IZ	
SP	

Operation size in words
(32 bits ← 16 bits × 16 bits)

RR	Code R
XWA	000
XBC	001
XDE	010
XHL	011
XIX	100
XIY	101
XIZ	110
XSP	111

rr of the MULS rr,# instruction is as listed below.

Operation size in bytes
(16 bits ← 8 bits × 8 bits)

rr	Code r
WA	001
BC	011
DE	101
HL	111
IX	C7H : F0H
IY	C7H : F4H
IZ	C7H : F8H
SP	C7H : FCH
	1st byte 2nd byte

Operation size in words
(32 bits ← 16 bits × 16 bits)

rr	Code r
XWA	000
XBC	001
XDE	010
XHL	011
XIX	100
XIY	101
XIZ	110
XSP	111

*1 Any other word registers can be specified in the same extension coding as those for IX to SP.

*2 Any other long word registers can be specified in the extension coding.

NEG dst

<Negate>

Operation: dst ← 0 – dst

Description: Decrements 0 by the contents of dst and loads the result to dst.
(Twos complement)

Details:

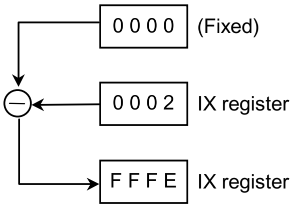
Size			Mnemonic		Code
Byte	Word	Long word			
○	○	×	NEG	r	1 1 0 z 1 r 0 0 0 0 0 1 1 1

Flags:

S	Z	H	V	N	C
*	*	*	*	1	*

S = MSB value of the result is set.
Z = 1 is set when the result is 0, otherwise 0.
H = 1 is set when a borrow from bit 3 to bit 4 occurs as a result, otherwise 0.
V = 1 is set when an overflow occurs as a result, otherwise 0.
N = 1 is set.
C = 1 is set when a borrow from the MSB occurs as a result, otherwise 0.

Execution example: NEG IX
When the IX register = 0002H, execution sets the IX register to FFFEH.



NOP

<No Operation>

Operation: None.

Description: Does nothing but moves execution to the next instruction. The object code of this instruction is 00H.

Details:

Mnemonic	Code
NOP	0 0 0 0 0 0 0 0

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change

OR dst, src

<Logical OR>

Operation: dst ← dst OR src

Description: Ors the contents of dst with those of src and loads the result to dst.

(Truth table)

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

Details:

	Size		Mnemonic		Code							
Byte	Word	Long word										
○	○	○	OR	R, r	1	1	z	z	1	r		
					1	1	1	0	0	R		
○	○	○	OR	r, #	1	1	z	z	1	r		
					1	1	0	0	1	1	1	0
					#<7:0>							
					#<15:8>							
					#<23:16>							
					#<31:24>							
○	○	○	OR	R, (mem)	1	m	z	z	m	m	m	m
					1	1	1	0	0	R		
○	○	○	OR	(mem), R	1	m	z	z	m	m	m	m
					1	1	1	0	1	R		
○	○	×	OR<W>	(mem), #	1	m	0	z	m	m	m	m
					0	0	1	1	1	1	1	0
					#<7:0>							
					#<15:8>							

Flags:

S	Z	H	V	N	C
*	*	0	*	0	0

S = MSB value of the result is set.
Z = 1 is set when the result is 0, otherwise 0.
H = 0 is set.
V = 1 is set when the parity (number of 1s) of the result is even, 0 when odd.
When the operand is 32-bit, an undefined value is set.
N = Cleared to 0.
C = Cleared to 0.

Execution example: OR HL, IX

When the HL register = 7350H and the IX register is 3456H, execution sets the HL register to 7756H.

```
       0111  0011  0101  0000 ← HL register (before execution)
OR)    0011  0100  0101  0110 ← IX register (before execution)
─────────────────────────────
       0111  0111  0101  0110 ← HL register (after execution)
```

ORCF num, src

<OR Carry Flag>

Operation: CY ← CY OR src<num>

Description: Ors the contents of the carry flag with those of bit num of src and loads the result to the carry flag.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	×	ORCF	#4, r	1 1 0 z 1 r 0 0 1 0 0 0 0 1 0 0 0 0 # 4
○	○	×	ORCF	A, r	1 1 0 z 1 r 0 0 1 0 1 0 0 1
○	×	×	ORCF	#3, (mem)	1 m 1 1 m m m m 1 0 0 0 1 #3
○	×	×	ORCF	A, (mem)	1 m 1 1 m m m m 0 0 1 0 1 0 0 1

Note: When bit num is specified by the A register, the value of the lower 4 bits of the A register is used as bit num. When the operand is a byte and the value of the lower bits of bit num is from 8 to 15, the result is undefined.

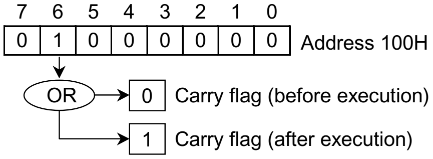
Flags:

S	Z	H	V	N	C
–	–	–	–	–	*

S = No change
Z = No change
H = No change
V = No change
N = No change
C = The result of or-ing the contents of the carry flag with those of bit num of src is set.

Execution example: ORCF 6, (100H)

When the contents of memory at address 100H = 01000000B (binary) and the carry flag = 0, execution sets the carry flag to 1.



PAA dst

<Pointer Adjust Accumulator>

Operation: if dst <LSB> = 1 then dst ← dst + 1

Description: Increments dst by 1 when the LSB of dst is 1. Does nothing when the LSB of dst is 0.

Used to make the contents of dst even. With the TLCS-900 series, when accessing 16- or 32-bit data in memory, if the data are loaded from an address starting with an even number, the number of bus cycles is 1 less than that of the data loaded from an address starting with an odd number.

Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
×	○	○	PAA	r	1 1 z z 1 r 0 0 0 1 0 1 0 0

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change

Execution example: PAA XIZ

When the XIZ register = 00234567H, execution increments the XIZ register by 1 so that it becomes 00234568H.

POP dst

<Pop>

Operation: dst ← (XSP+)

$$\left[\begin{array}{ll} \text{In bytes} & : \text{dst} \leftarrow \text{(XSP), XSP} \leftarrow \text{XSP} + 1 \\ \text{In words} & : \text{dst} \leftarrow \text{(XSP), XSP} \leftarrow \text{XSP} + 2 \\ \text{In long words} & : \text{dst} \leftarrow \text{(XSP), XSP} \leftarrow \text{XSP} + 4 \end{array}\right]$$

Description: First loads the contents of memory address specified by the stack pointer XSP to dst. Then increments the stack pointer XSP by the number of bytes in the operand.

Details:

	Size		Mnemonic		Code							
Byte	Word	Long word										
○	×	×	POP	F	0	0	0	1	1	0	0	1
○	×	×	POP	A	0	0	0	1	0	1	0	1
×	○	○	POP	R	0	1	0	s	1	R		
○	○	○	POP	r	1	1	z	z	1	r		
					0	0	0	0	0	1	0	1
○	○	×	POP<W>	(mem)	1	m	1	1	m	m	m	m
					0	0	0	0	0	1	z	0

Flags:

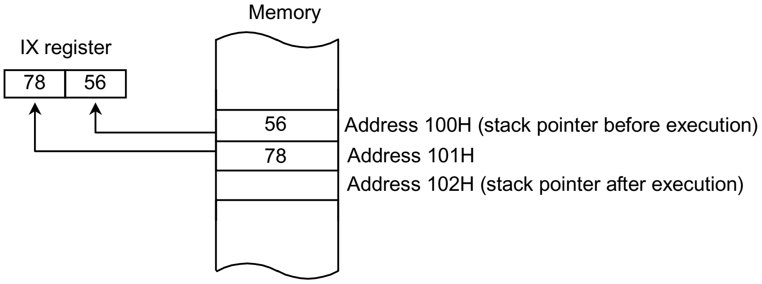
S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Note: Executing POP F changes all flags.

Execution example: POP IX

When the stack pointer XSP = 0100H, the contents of address 100H = 56H, and the contents of address 101H = 78H, execution sets the IX register to 7856H and the stack pointer XSP to 0102H.


Memory
IX register
78
56
56
78
Address 100H (stack pointer before execution)
Address 101H
Address 102H (stack pointer after execution)

POP SR

<Pop SR>

Operation: SR ← (XSP+)

Description: Loads the contents of the address specified by the stack pointer XSP to status register. Then increments the contents of the stack pointer XSP by 2.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
×	○	×	POP	SR	0 0 0 0 0 0 1 1

Flags:

S	Z	H	V	N	C
*	*	*	*	*	*

S =
Z =
H =
V =
N =
C =

} Contents of the memory address specified by the stack pointer XSP are set.

Note1: Please execute this instruction during DI condition.
The timing for executing this instruction is delayed by several states than that for fetching the instruction. This is because an instruction queue (4 bytes) and pipeline processing method is used.

Note2: The minimum mode is not supported for 900/L1. Therefor, the SR<MAX> register must be set to 1 by this instruction.

PUSH SR

<Push SR>

Operation: (–XSP) ← SR

Description: Decrements the contents of the stack pointer XSP by 2. Then loads the contents of status register to the memory address specified by the stack pointer XSP.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
×	○	×	PUSH	SR	0 0 0 0 0 0 1 0

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

PUSH src

<Push>

Operation: (–XSP) ← src

In bytes	: XSP ← XSP – 1, (XSP) ← src
In words	: XSP ← XSP – 2, (XSP) ← src
In long words	: XSP ← XSP – 4, (XSP) ← src

Description: Decrements the stack pointer XSP by the byte length of the operand. Then loads the contents of src to the memory address specified by the stack pointer XSP.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	×	×	PUSH	F	0 0 0 1 1 0 0 0
○	×	×	PUSH	A	0 0 0 1 0 1 0 0
×	○	○	PUSH	R	0 0 1 s 1 R
○	○	○	PUSH	r	1 1 z z 1 r 0 0 0 0 0 1 0 0
○	○	×	PUSH<W>	#	0 0 0 0 1 0 z 1 #<7:0> #<15:8>
○	○	×	PUSH<W>	(mem)	1 m 0 z m m m m 0 0 0 0 0 1 0 0

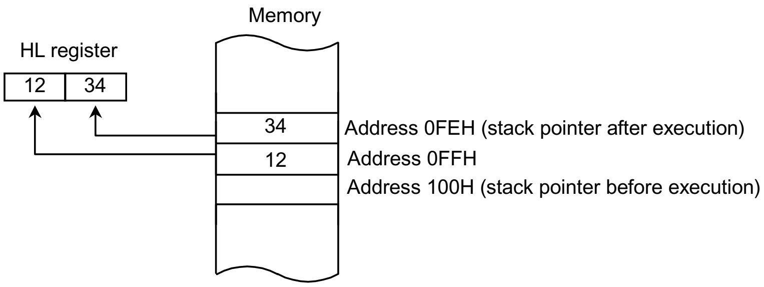
Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: PUSH HL

When the stack pointer XSP = 0100H and the HL register = 1234H, execution changes address 00FEH to 34H, address 00FFH to 12H, and sets the stack pointer XSP to 00FEH.


Memory
HL register
12
34
34
Address 0FEH (stack pointer after execution)
12
Address 0FFH
Address 100H (stack pointer before execution)

RCF
<Reset Carry Flag>

Operation: CY ← 0

Description: Resets the carry flag to 0.

Details:

Mnemonic	Code
RCF	0 0 0 1 0 0 0 0

Flags:

S	Z	H	V	N	C
–	–	0	–	0	0

S = No change
Z = No change
H = Reset to 0.
V = Reset to 0.
N = No change
C = Reset to 0.

RES num, dst

<Reset>

Operation: dst <num> ← 0

Description: Resets bit num of dst to 0.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	×	RES	#4, r	1 1 0 z 1 r 0 0 1 1 0 0 0 0 0 0 0 0 # 4
○	×	×	RES	#3, (mem)	1 m 1 1 m m m m 1 0 1 1 0 #3

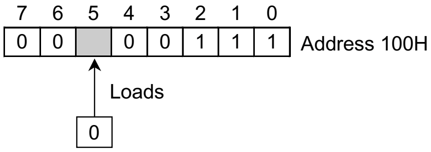
Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: RES 5, (100H)

When the contents of memory at address 100H = 00100111B (binary), execution sets the contents to 00000111B (binary).



RET condition

<Return>

Operation: If cc is true, then the 32-bit PC ← (XSP), XSP ← XSP + 4.

Description: Pops the return address from the stack area to the program counter when the operand condition is true.

Details:

Mnemonic		Code							
RET		0	0	0	0	1	1	1	0
RET	cc	1	0	1	1	0	0	0	0
		1	1	1	1		c	c	

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: RET

When the stack pointer XSP = 0FCH and the contents of memory at address 0FCH = 9000H (long word data), execution sets the stack pointer XSP to 100H and jumps (returns) to address 9000H.

RETD num

<Return and Deallocate>

Operation: 32-bit PC ← (XSP), XSP ← XSP + 4, XSP ← XSP + num

Description: Pops the return address from the stack area to the program counter. Then increments the stack pointer XSP by signed num.

Details:

Mnemonic		Code
RETD	d16	0 0 0 0 1 1 1 1
		d<7:0>
		d<15:8>

Flags:

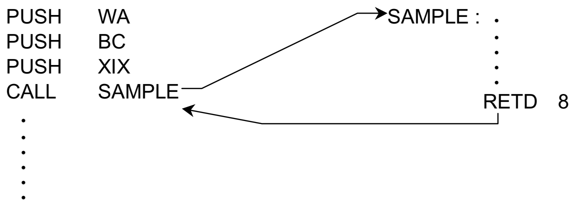
S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: RETD 8

When the stack pointer XSP = 0FCH and the contents of memory at address 0FCH = 9000H (long word data) in minimum mode, execution sets the stack pointer XSP to 0FCH + 4 + 8 → 108H and jumps (returns) to address 9000H.

Usage of the RETD instruction is shown below. In this example, the 8-bit parameter is pushed to the stack before the subroutine call. After the subroutine processing complete, the used parameter area is deleted by the RETD instruction.



```
PUSH    WA                    SAMPLE :  .
PUSH    BC                              .
PUSH    XIX                             .
CALL    SAMPLE                          .
   .                               RETD    8
   .
   .
   .
```

RETI

<Return from Interrupt>

Operation: SR ← (XSP), 32-bit PC ← (XSP + 2), XSP ← XSP + 6

After the above operation is executed, the 900/L1 decrement a value of interrupt nesting counter INTNEST by 1.

Description: Pops data from the stack area to status register and program counter.

After the above operation is executed, the 900/L1 decrement a value of interrupt nesting counter INTNEST by 1.

Details:

Mnemonic	Code
RETI	0 0 0 0 0 1 1 1

Flags:

S	Z	H	V	N	C
*	*	*	*	*	*

S = The value popped from the stack area is set.
Z = The value popped from the stack area is set.
H = The value popped from the stack area is set.
V = The value popped from the stack area is set.
N = The value popped from the stack area is set.
C = The value popped from the stack area is set.

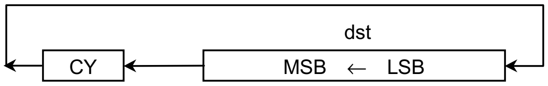
RL num, dst

<Rotate Left>

Operation: {CY & dst ← left rotates the value of CY & dst} Repeat num

Description: Rotates left the contents of the linked carry flag and dst.
Repeats the number of times specified in num.

Description figure:



Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	○	RL	#4, r	1 1 | z z | 1 | r 1 1 1 0 1 0 1 0 0 0 0 0 | # 4
○	○	○	RL	A, r	1 1 | z z | 1 | r 1 1 1 1 1 0 1 0
○	○	×	RL<W>	(mem)	1 | m | 0 | z | m m m m 0 1 1 1 1 0 1 0

Note: When the number of rotates is specified by the A register, the value of the lower 4 bits of the A register is used. Specifying 0 rotates 16 times.
When dst is memory, rotating is performed only once.

Flags:

S	Z	H	V	N	C
*	*	0	*	0	*

S = MSB value of dst after rotate is set.
Z = 1 is set when the contents of dst after rotate is 0, otherwise 0.
H = Reset to 0.
V = 1 is set when the parity (number of 1s) of dst is even after rotate, otherwise 0.
If the perand is 32 bits, an undefined value is set.
N = Reset to 0.
C = The value after rotate is set.

Execution example: RL 4, HL
When the HL register = 6230H and the carry flag = 1, execution sets the HL register to 230BH and the carry flag to 0.

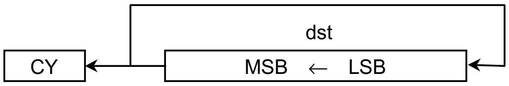
RLC num, dst

<Rotate Left without Carry>

Operation: {CY ← dst <MSB>, dst ← left rotate value of dst} Repeat num

Description: Loads the contents of the MSB of dst to the carry flag and rotates left the contents of dst. Repeats the number of times specified in num.

Description figure:



Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	○	RLC	#4, r	1 1 z z 1 r 1 1 1 0 1 0 0 0 0 0 0 0 # 4
○	○	○	RLC	A, r	1 1 z z 1 r 1 1 1 1 1 0 0 0
○	○	×	RLC<W>	(mem)	1 m 0 z m m m m 0 1 1 1 1 0 0 0

Note: When the number of rotates is specified by the A register, the value of the lower 4 bits of the A register is used. Specifying 0 rotates 16 times.
When dst is memory, rotating is performed only once.

Flags:

S	Z	H	V	N	C
*	*	0	*	0	*

S = MSB value of dst after rotate is set.
Z = 1 is set when the contents of dst after rotate is 0, otherwise, 0.
H = Reset to 0.
V = 1 is set when the parity (number of 1s) of dst is even after rotate.
If the operand is 32 bits, an undefined value is set.
N = Reset to 0.
C = MSB value of dst before the last rotate is set.

Execution example: RLC 4, HL
When the HL register = 1230H, execution sets the HL register to 2301H and the carry flag to 1.

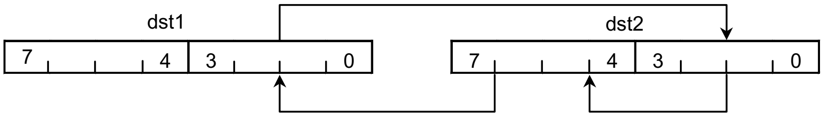
RLD dst1, dst2

<Rotate Left Digit>

Operation: dst1<3:0> ← dst2<7:4>, dst2<7:4> ← dst2<3:0>, dst2<3:0> ← dst1 <3:0>

Description: Rotates left the lower 4 bits of dst1 and the contents of dst2 in units of 4 bits.

Description figure:



Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	×	×	RLD	[A,] (mem)	1 m 0 0 m m m m 0 0 0 0 0 1 1 0

Flags:

S	Z	H	V	N	C
*	*	0	*	0	–

S = MSB value of the A register after rotate is set.
Z = 1 is set when the contents of the A register after the rotate are 0, otherwise 0.
H = Reset to 0.
V = 1 is set when the parity (number of 1s) of the A register is even after the rotate, otherwise 0.
N = Reset to 0.
C = No change

Execution example: RLD A, (100H)

When the A register = 12H and the contents of memory at address 100H = 34H, execution sets the A register to 13H and the contents of memory at address 100H to 42H.

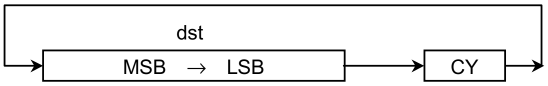
RR num, dst

<Rotate Right>

Operation: {CY & dst ← right rotates the value of CY & dst} Repeat num

Description: Rotates right the linked contents of the carry flag and dst.
Repeats the number of times specified in num.

Description figure:



Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	○	RR	#4, r	1 1 z z 1 r 1 1 1 0 1 0 1 1 0 0 0 0 # 4
○	○	○	RR	A, r	1 1 z z 1 r 1 1 1 1 1 0 1 1
○	○	×	RR<W>	(mem)	1 m 0 z m m m m 0 1 1 1 1 0 1 1

Note: When the number of rotates is specified by the A register, the value of the lower 4 bits of the A register is used. Specifying 0 rotates 16 times.
When dst is memory, rotating is performed only once.

Flags:

S	Z	H	V	N	C
*	*	0	*	0	*

S = MSB value of dst after rotate is set.
Z = 1 is set when the contents of dst after rotate is 0, otherwise 0.
H = Reset to 0.
V = 1 is set when the parity (number of 1s) of dst is even after the rotate, otherwise 0. If the operand is 32 bits, an undefined value is set.
N = Reset to 0.
C = The value after rotate is set.

Execution example: RR 4, HL
When the HL register = 6230H and the carry flag = 1, execution sets the HL register to 1623H and the carry flag to 0.

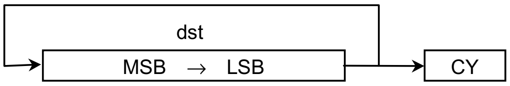
RRC num, dst

<Rotate Right without Carry>

Operation: {CY ← dst <LSB>, dst ← right rotate value of dst} Repeat num

Description: Loads the contents of the LSB of dst to the carry flag and rotates the contents of dst to the right. Repeats the number of times specified in num.

Description figure:



Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
○	○	○	RRC	#4, r	1 1 z z 1 r 1 1 1 0 1 0 0 1 0 0 0 0 # 4
○	○	○	RRC	A, r	1 1 z z 1 r 1 1 1 1 1 0 0 1
○	○	×	RRC<W>	(mem)	1 m 0 z m m m m 0 1 1 1 1 0 0 1

Note: When the number of rotates num is specified by the A register, the value of the lower 4 bits of the A register is used as the number of rotates.
Specifying 0 rotates 16 times. When dst is memory, rotating is only once.

Flags:

S	Z	H	V	N	C
*	*	0	*	0	*

S = MSB value of dst after rotate is set.
Z = 1 is set when the contents of dst after rotate is 0, otherwise 0.
H = Reset to 0.
V = 1 is set when the parity (number of 1s) of dst is even after rotate, otherwise 0.
If the operand is 32 bits, an undefined value is set.
N = Reset to 0.
C = MSB value of dst before the last rotate is set.

Execution example: RRC 4, HL
When the HL register = 1230H, execution sets the HL register to 0123H and the carry flag to 0.

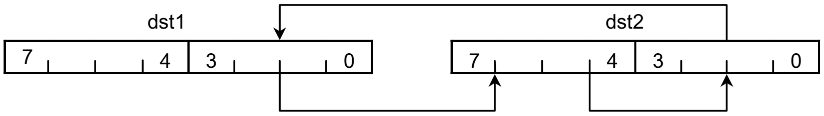
RRD dst1, dst2

<Rotate Right Digit>

Operation: dst1<3:0> ← dst2<3:0>, dst2<7:4> ← dst1<3:0>, dst2<3:0> ← dst2<7:4>

Description: Rotates right the lower 4 bits of dst1 and the contents of dst2 in units of 4 bits.

Description figure:



Details:

Size			Mnemonic		Code							
Byte	Word	Long word										
○	×	×	RRD	[A,](mem)	1	m	0	0	m	m	m	m
					0	0	0	0	0	1	1	1

Flags:

S	Z	H	V	N	C
*	*	0	*	0	–

S = MSB value of the A register after rotate is set.
Z = 1 is set when the contents of the A register after rotate is 0, otherwise 0.
H = Reset to 0.
V = 1 is set when the parity (number of 1s) of the A register is even after rotate, otherwise 0.
N = Reset to 0.
C = No change

Execution example: RRD A, (100H)

When the A register = 12H and the contents of memory at address 100H = 34H, execution sets the A register to 14H and the contents of memory at address 100H to 23H.

SBC dst, src

<Subtract with Carry>

Operation: dst ← dst – src – CY

Description: Subtracts the contents of src and the carry flag from those of dst, and loads the result to dst.

Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
○	○	○	SBC	R, r	1 1 z z 1 r 1 0 1 1 0 R
○	○	○	SBC	r, #	1 1 z z 1 r 1 1 0 0 1 0 1 1 #<7:0> #<15:8> #<23:16> #<31:24>
○	○	○	SBC	R, (mem)	1 m z z m m m m 1 0 1 1 0 R
○	○	○	SBC	(mem), R	1 m z z m m m m 1 0 1 1 1 R
○	○	×	SBC<W>	(mem), #	1 m 0 z m m m m 0 0 1 1 1 0 1 1 #<7:0> #<15:8>

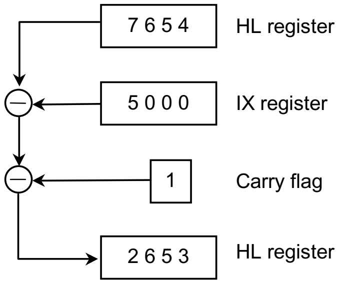
Flags:

S	Z	H	V	N	C
*	*	*	*	1	*

S = MSB value of the result is set.
Z = 1 is set when the result is 0, otherwise 0.
H = 1 is set when a borrow from bit 3 to bit 4 occurs as a result, otherwise 0.
When the operand is 32 bits, an undefined value is set.
V = 1 is set when an overflow occurs as a result, otherwise 0.
N = 1 is set.
C = 1 is set when a borrow from the MSB occurs as a result, otherwise 0.

Execution example: SBC HL, IX

When the HL register is 7654H, the IX register = 5000H, and the carry flag = 1, execution sets the HL register to 2653H.



SCC condition, dst

<Set Condition Code>

Operation: If cc is true, then dst ← 1 else dst ← 0.

Description: Loads 1 to dst when the operand condition is true; when false, 0 is loaded to dst.

Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
○	○	×	SCC	cc, r	1 1 0 z 1 r 0 1 1 1 c c

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: SCC OV, HL
When the contents of the V flag = 1, execution sets the HL register to 0001H.

SCF

<Set Carry Flag>

Operation: CY ← 1

Description: Sets the carry flag to 1.

Details:

Mnemonic	Code
SCF	0 0 0 1 0 0 0 1

Flags:

S	Z	H	V	N	C
–	–	0	–	0	1

S = No change
Z = No change
H = Reset to 0.
V = No change
N = Reset to 0.
C = Set to 1.

SET num, dst

<Set>

Operation: dst <num> ← 1

Description: Sets bit num of dst to 1.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	×	SET	#4, r	1 1 0 z 1 r 0 0 1 1 0 0 0 1 0 0 0 0 # 4
○	×	×	SET	#3, (mem)	1 m 1 1 m m m m 1 0 1 1 1 #3

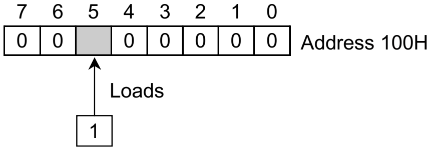
Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: SET 5, (100H)

When the contents of memory at address 100H = 00000000B (binary), execution sets the contents of memory at address 100H to 00100000B (binary).



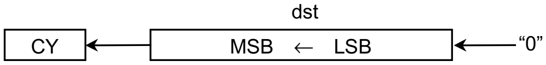
SLA num, dst

<Shift Left Arithmetic>

Operation: {CY ← dst<MSB>, dst ← left shift value of dst, dst<LSB> ← 0} Repeat num

Description: Loads the contents of the MSB of dst to the carry flag, shifts left the contents of dst, and loads 0 to the LSB of dst. Repeats the number of times specified in num.

Description chart:



Details:

	Size		Mnemonic		Code							
Byte	Word	Long word										
○	○	○	SLA	#4, r	1	1	z	z	1	r		
					1	1	1	0	1	1	0	0
					0	0	0	0	# 4			
○	○	○	SLA	A, r	1	1	z	z	1	r		
					1	1	1	1	1	1	0	0
○	○	×	SLA<W>	(mem)	1	m	0	z	m	m	m	m
					0	1	1	1	1	1	0	0

Note: When the number of shifts, num, is specified by the A register, the value of the lower 4 bits of the A register is used. Specifying 0 shifts 16 times. When dst is memory, shifting is performed only once.

Flags:

S	Z	H	V	N	C
*	*	0	*	0	*

S = MSB value of dst after shift is set.
Z = 1 is set when the contents of dst after shift is 0, otherwise 0.
H = Reset to 0.
V = 1 is set when the parity (number of 1s) of dst is even after shifting, otherwise 0.
If the operand is 32 bits, an undefined value is set.
N = Reset to 0.
C = MSB value of dst before the last shift is set.

Execution example: SLA 4, HL

When the HL register = 1234H, execution sets the HL register to 2340H and the carry flag to 1.

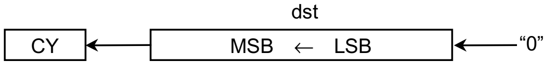
SLL num, dst

<Shift Left Logical>

Operation: {CY ← dst<MSB>, dst ← left shift value of dst, dst<LSB> ← 0} Repeat num

Description: Loads the contents of the MSB of dst to the carry flag, shifts left the contents of dst, and loads 0 to the MSB of dst. Repeats the number of times specified in num.

Description chart:



Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	○	SLL	#4, r	1 1 z z 1 r 1 1 1 0 1 1 1 0 0 0 0 0 # 4
○	○	○	SLL	A, r	1 1 z z 1 r 1 1 1 1 1 1 1 0
○	○	×	SLL<W>	(mem)	1 m 0 z m m m m 0 1 1 1 1 1 1 0

Note: When the number of shifts, num, is specified by the A register, the value of the lower 4 bits of the A register is used. Specifying 0 shifts 16 times. When dst is memory, shifting is performed only once.

Flags:

S	Z	H	V	N	C
*	*	0	*	0	*

S = MSB value of dst after shift is set.
Z = 1 is set when the contents of dst after shift is 0, otherwise 0.
H = Reset to 0.
V = 1 is set when the parity (number of 1s) of dst is even after shifting, otherwise 0.
If the operand is 32 bits, an undefined value is set.
N = Reset to 0.
C = MSB value of dst before the last shift is set.

Execution example: SLL 4, HL
When the HL register = 1234H, execution sets the HL register to 2340H and the carry flag to 1.

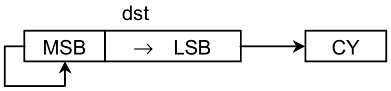
SRA num, dst

<Shift Right Arithmetic>

Operation: {CY ← dst<MSB>, dst ← right shift value of dst, dst <MSB> is fixed} Repeatnum

Description: Loads the contents of the LSB of dst to the carry flag and shifts right the contents of dst (MSB is fixed). Repeats the number of times specified in num.

Description chart:



Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	○	SRA	#4, r	1 1 z z 1 r
					1 1 1 0 1 1 0 0
					0 0 0 0 # 4
○	○	○	SRA	A, r	1 1 z z 1 r
					1 1 1 1 1 1 0 1
○	○	×	SRA<W>	(mem)	1 m 0 z m m m m
					0 1 1 1 1 1 0 1

Note: When the number of shifts, num, is specified by the A register, the value of the lower 4 bits of the A register is used. Specifying 0 shifts 16 times. When dst is memory, shifting is performed only once.

Flags:

S	Z	H	V	N	C
*	*	0	*	0	*

S = MSB value of dst after shift is set.
Z = 1 is set when the contents of dst after shift is 0, otherwise 0.
H = Reset to 0.
V = 1 is set when the parity (number of 1s) of dst is even after shift, otherwise 0.
If the operand is 32 bits, an undefined value is set.
N = Reset to 0.
C = LSB value of dst before the last shift is set.

Execution example: SRA 4, HL
When the HL register = 8230H, execution sets the HL register to F823H and the carry flag to 0.

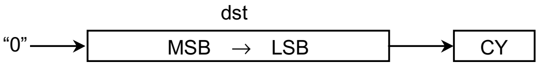
SRL num, dst

<Shift Right Logical>

Operation: {CY ← dst<LSB>, dst ← right shift value of dst, dst <MSB> ← 0} Repeat num

Description: Loads the contents of the LSB of dst to the carry flag, shifts right the contents of dst, and loads 0 to the MSB of dst. Repeats the number of times specified in num.

Description chart:



Details:

	Size		Mnemonic		Code							
Byte	Word	Long word										
○	○	○	SRL	#4, r	1	1	z	z	1	r		
					1	1	1	0	1	1	1	1
					0	0	0	0	# 4			
○	○	○	SRL	A, r	1	1	z	z	1	r		
					1	1	1	1	1	1	1	1
○	○	×	SRL<W>	(mem)	1	m	0	z	m	m	m	m
					0	1	1	1	1	1	1	1

Note: When the number of shifts, num, is specified by the A register, the value of the lower 4 bits of the A register is used. Specifying 0 shifts 16 times. When dst is memory, shifting is performed only once.

Flags:

S	Z	H	V	N	C
*	*	0	*	0	*

S = MSB value of dst after shift is set.
Z = 1 is set when the contents of dst after shift is 0, otherwise 0.
H = Reset to 0.
V = 1 is set when the parity (number of 1s) of dst is even after shift, otherwise 0.
If the operand is 32 bits, an undefined value is set.
N = Reset to 0.
C = LSB value of dst before the last shift is set.

Execution example: SRL 4, HL

When the HL register = 1238H, execution sets the HL register to 0123H and the carry flag to 1.

STCF num, dst

<Store Carry Flag>

Operation: dst<num> ← CY

Description: Loads the contents of the carry flag to bit num of dst.

Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
○	○	×	STCF	#4, r	1 1 0 z 1 r 0 0 1 0 0 1 0 0 0 0 0 0 # 4
○	○	×	STCF	A, r	1 1 0 z 1 r 0 0 1 0 1 1 0 0
○	×	×	STCF	#3, (mem)	1 m 1 1 m m m m 1 0 1 0 0 #3
○	×	×	STCF	A, (mem)	1 m 1 1 m m m m 0 0 1 0 1 1 0 0

Note: When bit num is specified by the A register, the value of the lower 4 bits of the A register is used. When the operand is a byte and the value of the lower 4 bits of bit num is from 8 to 15, the operand value does not change.

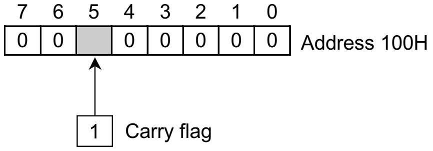
Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: STCF 5, (100H)

When the contents of memory at address 100H = 00H and the carry flag = 1, execution sets the contents of memory at address 100H to 00100000B (binary).



SUB dst, src

<Subtract>

Operation: dst ← dst − src

Description: Subtracts the contents of src from those of dst and loads the result to dst.

Details:

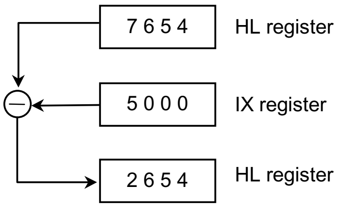
	Size		Mnemonic		Code							
Byte	Word	Long word										
○	○	○	SUB	R, r	1	1	z	z	1	r		
					1	0	1	0	0	R		
○	○	○	SUB	r, #	1	1	z	z	1	r		
					1	1	0	0	1	0	1	0
					#<7:0>							
					#<15:8>							
					#<23:16>							
					#<31:24>							
○	○	○	SUB	R, (mem)	1	m	z	z	m	m	m	m
					1	0	1	0	0	R		
○	○	○	SUB	(mem), R	1	m	z	z	m	m	m	m
					1	0	1	0	1	R		
○	○	×	SUB<W>	(mem), #	1	m	0	z	m	m	m	m
					0	0	1	1	1	0	1	0
					#<7:0>							
					#<15:8>							

Flags:

S	Z	H	V	N	C
*	*	*	*	1	*

S = MSB value of the result is set.
Z = 1 is set when the result is 0, otherwise 0.
H = 1 is set when a borrow from bit 3 to bit 4 occurs as a result, otherwise 0.
When the operand is 32 bits, an undefined value is set.
V = 1 is set when an overflow occurs as a result, otherwise 0.
N = 1 is set.
C = 1 is set when a borrow from MSB occurs as a result, otherwise 0.

Execution example: SUB HL, IX
When the HL register = 7654H and the IX register = 5000H, execution sets the HL register to 2654H.



SWI num

<Software Interrupt>

Operation:
1) XSP ← XSP − 6
2) (XSP) ← SR
3) (XSP + 2) ← 32 bit PC
4) PC ← (Address refer to vector + num × 4)

Note: Address refer to vector is defined for each product.

Description: Saves to the stack area the contents of the status register and contents of the program counter which indicate the address next to the SWI instruction. Finally, jumps to vector is indicated address refer to vector.

Details:

Size	Mnemonic		Code
	SWI	[#3]	1 1 1 1 1 | #3

Note 1: A value from 0 to 7 can be specified as the operand value. When the operand coding is omitted, SWI 7 is assumed.

Note 2: The status register structure is as shown below.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SYSM	IFF2	IFF1	IFF0	MAX	RFP2	RFP1	RFP0	S	Z	0	H	0	V	N	C

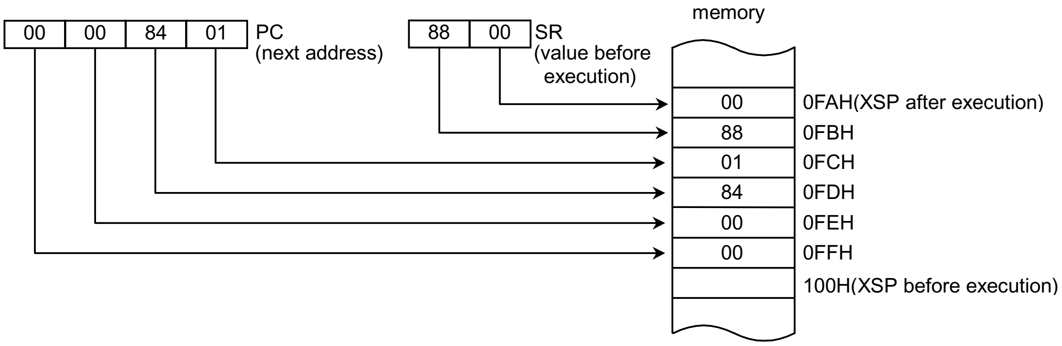
Flags:

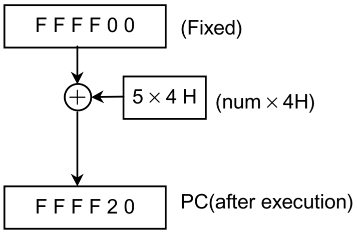
S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: SWI 5

When the stack pointer XSP = 100H, the status register = 8800H, executing the above instruction at memory address 8400H writes the contents of the previous status register 8800H in memory address 00FAH, and the contents of the program counter 00008401H in memory address 00FCH, then jumps to address FFFF20H.





TSET num, dst

<Test and Set>

Operation: Z flag ← inverted value of dst <num>
dst <num> ← 1

Description: Loads the inverted value of the bit num of dst to the Z flag.
Then the bit num of dst is set to 1.

Details:

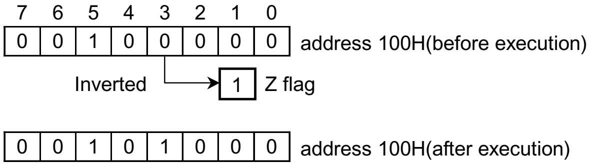
Size			Mnemonic		Code
Byte	Word	Long word			
○	○	×	TSET	#4, r	1 1 z z 1 r 0 0 1 1 0 1 0 0 0 0 0 0 # 4
○	×	×	TSET	#3, (mem)	1 m 1 1 m m m m 1 0 1 0 1 #3

Flags:

S	Z	H	V	N	C
×	*	1	×	0	–

S = An undefined value is set.
Z = The inverted value of the src <num> is set.
H = Set to 1
V = An undefined value is set.
N = Set to 0
C = No change

Execution example: When the contents of memory at address 100H = 00100000B (binary), TSET 3, (100H) execution sets the Z flag to 1, the contents of memory at address 100H = 00101000B (binary).



UNLK dst

<Unlink>

Operation: XSP ← dst, dst ← (XSP+)

Description: Loads the contents of dst to the stack pointer XSP, then pops long word data from the stack area to dst. Used paired with the Link instruction.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
×	×	○	UNLK	r	1 1 1 0 1 r 0 0 0 0 1 1 0 1

Flags:

S	Z	H	V	N	C
–	–	–	–	–	–

S = No change
Z = No change
H = No change
V = No change
N = No change
C = No change

Execution example: UNLK XIZ

As a result of executing this instruction after executing the Link instruction, the stack pointer XSP and the XIZ register revert to the same values they had before the Link instruction was executed. (For details of the Link instruction, see page 101)

XOR dst, src

<Exclusive OR>

Operation: dst ← dst XOR src

Description: Exclusive ors the contents of dst with those of src and loads the result to dst.

(Truth table)

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Details:

	Size		Mnemonic		Code
Byte	Word	Long word			
○	○	○	XOR	R, r	1 1 z z 1 r 1 1 0 1 0 R
○	○	○	XOR	r, #	1 1 z z 1 r 1 1 0 0 1 1 0 1 #<7:0> #<15:8> #<23:16> #<31:24>
○	○	○	XOR	R, (mem)	1 m z z m m m m 1 1 0 1 0 R
○	○	○	XOR	(mem), R	1 m z z m m m m 1 1 0 1 1 R
○	○	×	XOR<W>	(mem), #	1 m 0 z m m m m 0 0 1 1 1 1 0 1 #<7:0> #<15:8>

Flags:

S	Z	H	V	N	C
*	*	0	*	0	0

S = MSB value of the result is set.
Z = 1 is set when the result is 0, otherwise 0.
H = Reset to 0.
V = 1 is set when the parity (number of 1s) of dst is even as a result, otherwise 0.
If the operand is 32 bits, an undefined value is set.
N = Cleared to 0.
C = Cleared to 0.

Execution example: XOR HL, IX

When the HL register = 7350H and the IX register = 3456H, execution sets the HL register to 4706H.

```
        0111  0011  0101  0000 ← HL register (before execution)
XOR)    0011  0100  0101  0110 ← IX register (before execution)
----------------------------
        0100  0111  0000  0110 ← HL register (after execution)
```

XORCF num, src

<Exclusive OR Carry Flag>

Operation: CY ← CY XOR src<num>

Description: Exclusive ors the contents of the carry flag and bit num of src, and loads the result to the carry flag.

Details:

Size			Mnemonic		Code
Byte	Word	Long word			
○	○	×	XORCF	#4, r	1 1 0 z 1 r 0 0 1 0 0 0 1 0 0 0 0 0 # 4
○	○	×	XORCF	A, r	1 1 0 z 1 r 0 0 1 0 1 0 1 0
○	×	×	XORCF	#3, (mem)	1 m 1 1 m m m m 1 0 0 1 0 #3
○	×	×	XORCF	A, (mem)	1 m 1 1 m m m m 0 0 1 0 1 0 1 0

Note: When bit num is specified by the A register, the value of the lower 4 bits of the A register is used. When the operand is a byte and the value of the lower 4 bits of bit num is from 8 to 15, the result is undefined.

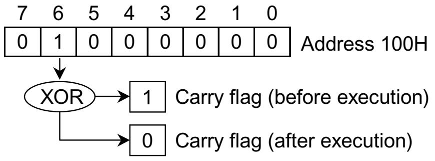
Flags:

S	Z	H	V	N	C
–	–	–	–	–	*

S = No change
Z = No change
H = No change
V = No change
N = No change
C = The value obtained by exclusive or-ing the contents of the carry flag with those of bit num of src is set.

Execution example: XORCF 6, (100H)

When the contents of memory at address 100H = 01000000B (binary) and the carry flag = 1, execution sets the carry flag to 0.



ZCF

<Zero flag to Carry Flag>

Operation: CY ← inverted value of Z flag

Description: Loads the inverted value of the Z flag to the carry flag.

Details:

Mnemonic	Code
ZCF	0 0 0 1 0 0 1 1

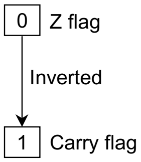
Flags:

S	Z	H	V	N	C
–	–	×	–	0	*

S = No change
Z = No change
H = An undefined value is set.
V = No change
N = Reset to 0.
C = The inverted value of the Z flag is set.

Execution example: ZCF

When the Z flag = 0, execution sets the carry flag to 1.



Appendix B Instruction Lists

- Explanation of symbols used in this document

1. Size

B	The operand size is in bytes (8 bits)
W	The operand size is in word (16 bits)
L	The operand size is in long word (32 bits)

2. Mnemonic

R	Eight general-purpose registers including 8/16/32-bit current bank registers. 8 bit register: W, A, B, C, D, E, H, L 16 bit register: WA, BC, DE, HL, IX, IY, IZ, SP 32 bit register: XWA, XBC, XDE, XHL, XIX, XIY, XIZ, XSP
r	8/16/32-bit registers
cr	All 8/16/32-bit CPU control registers DMAS0 to 3, DMAD0 to 3, DMAC0 to 3, DMAM0 to 3, INTNEST
A	A register (8 bits)
F	Flag registers (8 bits)
F’	Inverse flag registers (8 bits)
SR	Status registers (16 bits)
PC	Program Counter (in minimum mode,16 bits; in maximum mode, 32 bits)
(mem)	8/16/32-bit memory data
mem	Effective address value
<W>	When the operand size is a word, W must be specified.
[ ]	Operands enclosed in square brackets can be omitted.
#	8/16/32-bit immediate data.
#3	3-bit immediate data: 0 to 7 or 1 to 8 ⋯⋯ for abbreviated codes.
#4	4-bit immediate data: 0 to 15 or 1 to 16
d8	8-bit displacement: −80H to + 7FH
d16	16-bit displacement: −8000H to + 7FFFH
cc	Condition code
(#8)	Direct addressing : (00H) to (0FFH) ⋯ 256-byte area
(#16)	64K-byte area addressing : (0000H) to (0FFFFH)
$	A start address of the instruction is located

3. Code

Z	The code crepresent the operand sizes. byte (8 bit) = 0 word (16 bit) = 2 long word (32 bit) = 4
ZZ	The code represent the operand sizes. byte (8 bit) = 00H word (16 bit) = 10H long word (32 bit) = 20H

4. Flag (SZHVNC)

—	Flag doesn't change.
*	Flag changes by executing instruction.
0	Flag is cleared to 0.
1	Flag is set to 1.
P	Flag changes by executing instruction (It works as parity flag).
V	Flag changes by executing instruction (It works as overflow flag).
X	An undefined value is set in flag.

5. Instruction length

Instruction length is represented in byte unit.

+#	adds immediate data length.
+M	adds addressing code length.
+#M	adds immediate data length and addressing code length.

6. State

Execution processing time of instruction are shown in order of 8 bit, 16 bit, 32 bit processing in status unit.

1 state = 2 × 1/f_{FPH}

- 900/L1 Instruction Lists (1/10)

(1) Load

Group	Size	Mnemonic	Codes (16 hex)	Function	SZHVNC	Length (byte)	State
LD	BWL	LD R, r	C8 + zz + r : 88 + R	R ← r	––––––	2	2. 2. 2
	BWL	LD r, R	C8 + zz + r : 98 + R	r ← R	––––––	2	2. 2. 2
	BWL	LD r, #3	C8 + zz + r : A8 + #3	r ← #3	––––––	2	2. 2. 2
	BWL	LD R, #	20 + zz + R : #	R ← #	––––––	1 + #	2. 3. 5
	BWL	LD r, #	C8 + zz + r : 03 : #	r ← #	––––––	2 + #	3. 4. 6
	BWL	LD R, (mem)	80 + zz + mem : 20 + R	R ← (mem)	––––––	2 + M	4. 4. 6
	BWL	LD (mem), R	B0 + mem : 40 + zz + R	(mem) ← R	––––––	2 + M	4. 4. 6
	BW–	LD<W> (#8) , #	08 + z : #8 : #	(#8) ← #	––––––	2 + #	5. 6. –
	BW–	LD<W> (mem), #	B0 + mem : 00 + z : #	(mem) ← #	––––––	2 + M#	5. 6. –
	BW–	LD<W> (#16), (mem)	80 + zz + mem : 19 : #16	(#16) ← (mem)	––––––	4 + M	8. 8. –
	BW–	LD<W> (mem), (#16)	B0 + mem : 14 + z : #16	(mem) ← (#16)	––––––	4 + M	8. 8. –
PUSH	B––	PUSH F	18	(–XSP) ← F	––––––	1	3. –. –
	B––	PUSH A	14	(–XSP) ← A	––––––	1	3. –. –
	–WL	PUSH R	18 + zz + R	(–XSP) ← R	––––––	1	–. 3. 5
	BWL	PUSH r	C8 + zz + r : 04	(–XSP) ← r	––––––	2	4. 4. 6
	BW–	PUSH<W> #	09 + z : #	(–XSP) ← #	––––––	1 + #	4. 5. –
	BW–	PUSH<W> (mem)	80 + zz + mem : 04	(–XSP) ← (mem)	––––––	2 + M	6. 6. –
POP	B––	POP F	19	F ← (XSP+)	* * * * * *	1	4. –. –
	B––	POP A	15	A ← (XSP+)	––––––	1	4. –. –
	–WL	POP R	38 + zz + R	R ← (XSP+)	––––––	1	–. 4. 6
	BWL	POP r	C8 + zz + r : 05	r ← (XSP+)	––––––	2	5. 5. 7
	BW–	POP<W>(mem)	B0 + em : 04 + z	(mem) ← (XSP+)	––––––	2 + M	7. 7. –
LDA	–WL	LDA R, mem	B0 + mem : 10 + zz + R	R ← mem	––––––	2 + M	–. 4. 4
LDAR	–WL	LDAR R, $ + 4 + d16	F3:13:d16 : 20 + zz + R	R ← PC + d16	––––––	5	–. 7. 7

(2) Exchange

Group	Size	Mnemonic	Codes (16 hex)	Function	SZHVNC	Length (byte)	State
EX	B––	EX F, F'	16	F ↔ F'	* * * * * *	1	2. –. –
	BW–	EX R, r	C8 + zz + r : B8 + R	R ↔ r	––––––	2	3. 3. –
	BW–	EX (mem), R	80 + zz + mem : 30 + R	(mem) ↔ R	––––––	2 + M	6. 6. –
MIRR	–W–	MIRR r	D8 + r : 16	r<0:MSB> ← r<MSB : 0>	––––––	2	–. 3. –

- 900/L1 Instruction Lists (2/10)

(3) Load/Increment/Decrement & Compare Increment/Decrement Size

Group	Size	Mnemonic	Codes (16 hex)	Function	SZHVNC	Length (byte)	State
LDxx	BW−	LDI<W> [(XDE+), (XHL+)]	83 + zz : 10	(XDE+) ← (XHL+) BC ← BC − 1	−−0 0−	2	8. 8. −
	BW−	LDI<W> (XIX+), (XIY+)	85 + zz : 10	(XIX+) ← (XIY+) BC ← BC − 1	−−0 0−	2	8. 8. −

	BW−	LDIR<W>
		[(XDE+), (XHL+)]
			83 + zz : 11	repeat
				(XDE+) ← (XHL+)
				BC ← BC − 1
				until BC = 0
					−−000−	2	7n + 1

	BW−	LDIR<W>
		(XIX+), (XIY+)
			85 + zz : 11	repeat
				(XIX+) ← (XIY+)
				BC ← BC − 1
				until BC = 0
					−−000−	2	7n + 1

	BW−	LDD<W> [(XDE−), (XHL−)]	83 + zz : 12	(XDE−) ← (XHL−) BC ← BC − 1	−−0 0−	2	8. 8. −
	BW−	LDD<W> (XIX−), (XIY-)	85 + zz : 12	(XIX−) ← (XIY−) BC ← BC − 1	−−0 0−	2	8. 8. −

	BW−	LDDR<W>
		[(XDE-), (XHL-)]
			83 + zz : 13	repeat
				(XDE−) ← (XHL−)
				BC ← BC − 1
				until BC = 0
					−−000−	2	7n + 1

	BW−	LDDR<W>
		(XIX−), (XIY−)
			85 + zz : 13	repeat
				(XIX−) ← (XIY−)
				BC ← BC − 1
				until BC = 0
					−−000−	2	7n + 1

CPxx	BW−	CPI [A/WA, (R+)]	80 + zz + R : 14	A/WA − (R+) BC ← BC − 1	* * 1−	2	6. 6. −
	BW−	CPIR [A/WA, (R+)]	80 + zz + R : 15	repeat A/WA − (R+) BC ← BC − 1 until A/WA = (R) or BC=0	* * 1−	2	6n + 1
	BW−	CPD [A/WA, (R−)]	80 + zz + R : 16	A/WA − (R−) BC ← BC − 1	* * 1−	2	6. 6. −
	BW−	CPDR [A/WA, (R−)]	80 + zz + R : 17	repeat A/WA − (R−) BC ← BC − 1 until A/WA = (R) or BC = 0	* * 1−	2	6n + 1

Note 1: ①; If BC = 0 after execution, the P/V flag is set to 0, otherwise 1.

②; If A/WA = (R), the Z flag is set to 1, otherwise, 0 is set.

Note 2: When the operand is omitted in the CPI, CPIR, CPD, or CPDR instruction, A, (XHL+/−) is used as the default value.

- 900/L1 Instruction Lists (3/10)

(4) Arithmetic Operations

Group	Size	Mnemonic	Codes (16 hex)	Function	SZHVNC	Length (byte)	State
ADD	BWL	ADD R, r	C8 + zz + r : 80 + R	R ← R + r	***V0*	2	2. 2. 2
	BWL	ADD r, #	C8 + zz + r : C8 : #	r ← r + #	***V0*	2 + #	3. 4. 6
	BWL	ADD R, (mem)	80 + zz + mem : 80 + R	R ← R + (mem)	***V0*	2 + M	4. 4. 6
	BWL	ADD (mem), R	80 + zz + mem : 88 + R	(mem) ← (mem) + R	***V0*	2 + M	6. 6. 10
	BW–	ADD<W> (mem), #	80 + zz + mem : 38 : #	(mem) ← (mem) + #	***V0*	2 + M#	7. 8. –
ADC	BWL	ADC R, r	C8 + zz + r : 90 + R	R ← R + r + CY	***V0*	2	2. 2. 2
	BWL	ADC r, #	C8 + zz + r : C9 : #	r ← r + # + CY	***V0*	2 + #	3. 4. 6
	BWL	ADC R, (mem)	80 + zz + mem : 90 + R	R ← R + (mem) + CY	***V0*	2 + M	4. 4. 6
	BWL	ADC (mem), R	80 + zz + mem : 98 + R	(mem) ← (mem) + R + CY	***V0*	2 + M	6. 6. 10
	BW–	ADC<W> (mem), #	80 + zz + mem : 39 : #	(mem) ← (mem) + # + CY	***V0*	2 + M#	7. 8. –
SUB	BWL	SUB R, r	C8 + zz + r : A0 + R	R ← R – r	***V1*	2	2. 2. 2
	BWL	SUB r, #	C8 + zz +r : CA : #	r ← r – #	***V1*	2 + #	3. 4. 6
	BWL	SUB R, (mem)	80 + zz + mem : A0 + R	R ← R – (mem)	***V1*	2 + M	4. 4. 6
	BWL	SUB (mem), R	80 + zz + mem : A8 + R	(mem) ← (mem) – R	***V1*	2 + M	6. 6. 10
	BW–	SUB<W> (mem), #	80 + zz + mem : 3A : #	(mem) ← (mem) – #	***V1*	2 + M#	7. 8. –
SBC	BWL	SBC R, r	C8 + zz + r : B0 + R	R ← R – r – CY	***V1*	2	2. 2. 2
	BWL	SBC r, #	C8 + zz + r : CB : #	r ← r – # – CY	***V1*	2 + #	3. 4. 6
	BWL	SBC R, (mem)	80 + zz + mem : B0 + R	R ← R – (mem) – CY	***V1*	2 + M	4. 4. 6
	BWL	SBC (mem), R	80 + zz + mem : B8 + R	(mem) ← (mem) – R – CY	***V1*	2 + M	6. 6. 10
	BW–	SBC<W> (mem), #	80 + zz + mem : 3B : #	(mem) ← (mem) – # – CY	***V1*	2 + M#	7. 8. –
CP	BWL	CP R, r	C8 + zz + r : F0 + R	R – r	***V1*	2	2. 2. 2
	BW–	CP r, #3	C8 + zz + r : D8 + #3	r – #3	***V1*	2	2. 2. –
	BWL	CP r, #	C8 + zz + r : CF : #	r – #	***V1*	2 + #	3. 4. 6
	BWL	CP R, (mem)	80 + zz + mem : F0 + R	R – (mem)	***V1*	2 + M	4. 4. 6
	BWL	CP (mem), R	80 + zz + mem : F8 + R	(mem) – R	***V1*	2 + M	4. 4. 6
	BW–	CP<W> (mem), #	80 + zz + mem : 3F : #	(mem) – #	***V1*	2 + M#	5. 6. –
INC	B––	INC #3, r	C8 + r : 60 + #3	r ← r + #3	***V0–	2	2. –. –
	–WL	INC #3, r	C8 + zz + r : 60 + #3	r ← r + #3	––––––	2	–. 2. 2
	BW–	INC<W> #3, (mem)	80 + zz + mem : 60 + #3	(mem) ← (mem) + #3	***V0–	2 + M	6. 6. –
DEC	B––	DEC #3, r	C8 + r : 68 + #3	r ← r – #3	***V1–	2	2. –. –
	–WL	DEC #3, r	C8 + zz + r : 68 + #3	r ← r – #3	––––––	2	–. 2. 2
	BW–	DEC<W> #3, (mem)	80 + zz + mem : 68 + #3	(mem) ← (mem) – #3	***V1–	2 + M	6. 6. –
NEG	BW–	NEG r	C8 + zz + r : 07	r ← 0 – r	***V1*	2	2. 2. –
EXTZ	–WL	EXTZ r	C8 + zz + r : 12	r<high> ← 0	––––––	2	–. 3. 3
EXTS	–WL	EXTS r	C8 + zz + r : 13	r<high> ← r<low. MSB>	––––––	2	–. 3. 3
DAA	B––	DAA r	C8 + r : 10	Decimal adjustment after addition or subtraction	***P–*	2	4. –. –
PAA	–WL	PAA r	C8 + zz + r : 14	if r<0> = 1 then INC r	––––––	2	–. 4. 4

Note 1: With the INC/DEC instruction, when the code value of #3 = 0, functions as +8/–8.

Note 2: When the ADD R, r (word type) instruction is used in the TLCS-90, the S, Z, and V flags do not change. In the TLCS-900, these flags change.

- 900/L1 Instruction Lists (4/10)

Group	Size	Mnemonic	Codes (16 hex)	Function	SZHVNC	Length (byte)	State
MUL	BW–	MUL RR, r	C8 + zz + r : 40 + R	RR ← R × r	––––––	2	11.14. –
	BW–	MUL rr, #	C8 + zz + r : 08 : #	rr ← r × #	––––––	2 + #	12.15. –
	BW–	MUL RR, (mem)	80 + zz + mem: 40 + R	RR ← R × (mem)	––––––	2 + M	13.16. –
MULS	BW–	MULS RR, r	C8 + zz + r : 48 + R	RR ← R × r ;signed	––––––	2	9.12. –
	BW–	MULS rr, #	C8 + zz + r : 09 : #	rr ← r × # ;signed	––––––	2 + #	10.13. –
	BW–	MULS RR, (mem)	80 + zz + mem : 48 + R	RR ←R × (mem);signed	––––––	2 + M	11.14. –
DIV	BW–	DIV RR, r	C8 + zz + r : 50 + R	R ← RR ÷ r	–––V––	2	15.23. –
	BW–	DIV rr, #	C8 + zz + r : 0A : #	r ← rr ÷ #	–––V––	2 + #	15.23. –
	BW–	DIV RR, (mem)	80 + zz + mem : 50 + R	R ← RR ÷ (mem)	–––V––	2 + M	16.24. –
DIVS	BW–	DIVS RR, r	C8 + zz + r : 58 + R	R ← RR ÷ r ;signed	–––V––	2	18.26. –
	BW–	DIVS rr, #	C8 + zz + r : 0B : #	r ← rr ÷ # ;signed	–––V––	2 + #	18.26. –
	BW–	DIVS RR, (mem)	80 + zz + mem : 58 + R	R ← RR ÷ (mem);signed	–––V––	2 + M	19.27. –
MULA	–W–	MULA rr	D8 + r : 19	Multiply and add signed $\underline{rr} \leftarrow \underline{rr} + \underline{(XDE)} \times \underline{(XHL)}$ 32 bit 32 bit 16 bit 16 bit XHL ← XHL–2	**–V––	2	–.19. –

MINC	–W–	MINC1 #, r
		(# = 2**n)
		(1< = n< = 15)
			D8 + r : 38 : # – 1	modulo increment ;+1
				if (r mod #) = (# – 1)
				then r ← r – (# – 1)
				else r ← r + 1
					––––––	4	–. 5. –

	–W–	MINC2 #, r
		(# = 2**n)
		(2< = n< = 15)
			D8 + r : 39 : # – 2	modulo increment ;+2
				if (r mod #) = (# – 2)
				then r ← r – (# – 2)
				else r ← r + 2
					––––––	4	–. 5. –

	–W–	MINC4 #, r
		(# = 2**n)
		(3< = n< = 15)
			D8 + r : 3A : # – 4	modulo increment ;+4
				if (r mod #) = (# – 4)
				then r ← r – (# – 4)
				else r ← r + 4
					––––––	4	–. 5. –

MDEC	–W–	MDEC1 #, r
		(# = 2**n)
		(1< = n< = 15)
			D8 + r : 3C : # – 1	modulo decrement ;–1
				if (r mod #) = 0
				then r ← r + (# – 1)
				else r ← r – 1
					––––––	4	–. 4. –

	–W–	MDEC2 #, r
		(# = 2**n)
		(2< = n< = 15)
			D8 + r : 3D : # – 2	modulo decrement ;–2
				if (r mod #) = 0
				then r ← r + (# – 2)
				else r ← r – 2
					––––––	4	–. 4. –

	–W–	MDEC4 #, r
		(# = 2**n)
		(3< = n< = 15)
			D8 + r : 3E : # – 4	modulo decrement ;–4
				if (r mod #) = 0
				then r ← r + (# – 4)
				else r ← r – 4
					––––––	4	–. 4. –

Note: Operand RR of the MUL, MULS, DIV, and DIVS instructions indicates that a register twice the size of the operation is specified. When the operation is in bytes (8 bits × 8 bits, 16/8 bits), word register (16 bits) is specified; when the operation is in words (16 bits × 16 bits, 32/16 bits), long word register (32 bits) is specified.

- 900/L1 Instruction Lists (5/10)

(5) Logical operations

Group	Size	Mnemonic	Codes (16 hex)	Function	SZHVNC	Length (byte)	State
AND	BWL	AND R, r	C8 + zz + r : C0 + R	R ← R and r	**1P00	2	2. 2. 2
	BWL	AND r, #	C8 + zz + r : CC : #	r ← r and #	**1P00	2 + #	3. 4. 6
	BWL	AND R, (mem)	80 + zz + mem : C0 + R	R ← R and (mem)	**1P00	2 + M	4. 4. 6
	BWL	AND (mem), R	80 + zz + mem : C8 + R	(mem) ← (mem) and R	**1P00	2 + M	6. 6. 10
	BW−	AND<w> (mem), #	80 + zz + mem : 3C : #	(mem) ← (mem) and #	**1P00	2 + M#	7. 8. −
OR	BWL	OR R, r	C8 + zz + r : E0 + R	R ← R or r	**0P00	2	2. 2. 2
	BWL	OR r, #	C8 + zz + r : CE : #	r ← r or #	**0P00	2 + #	3. 4. 6
	BWL	OR R, (mem)	80 + zz + mem : E0 + R	R ← R or (mem)	**0P00	2 + M	4. 4. 6
	BWL	OR (mem), R	80 + zz + mem : E8 + R	(mem) ← (mem) or R	**0P00	2 + M	6. 6. 10
	BW−	OR<W> (mem), #	80 + zz + mem : 3E : #	(mem) ← (mem) or #	**0P00	2 + M#	7. 8. −
XOR	BWL	XOR R, r	C8 + zz + r : D0 + R	R ← R xor r	**0P00	2	2. 2. 2
	BWL	XOR r, #	C8 + zz + r : CD : #	r ← r xor #	**0P00	2 + #	3. 4. 6
	BWL	XOR R, (mem)	80 + zz + mem : D0 + R	R ← R xor (mem)	**0P00	2 + M	4. 4. 6
	BWL	XOR (mem), R	80 + zz + mem : D8 + R	(mem) ← (mem) xor R	**0P00	2 + M	6. 6. 10
	BW−	XOR<W> (mem), #	80 + zz + mem : 3D : #	(mem) ← (mem) xor #	**0P00	2 + M#	7. 8. −
CPL	BW−	CPL r	C8 + zz + r : 06	r ← not r	−−1−1−	2	2. 2. −

- 900/L1 Instruction Lists (6/10)

(6) Bit operations

Group	Size	Mnemonic	Codes (16 hex)	Function	SZHVNC	Length (byte)	State
LDCF	BW–	LDCF #4, r	C8 + zz + r : 23 : #4	CY ← r<#4>	−−−−−*	3	3.3.–
	BW–	LDCF A , r	C8 + zz + r : 2B	CY ← r<A>	−−−−−*	2	3.3.–
	B––	LDCF #3, (mem)	B0 + men : 98 + #3	CY ← (mem)<#3>	−−−−−*	2 + M	6.–.–
	B––	LDCF A , (mem)	B0 + men : 2B	CY ← (mem)<A>	−−−−−*	2 + M	6.–.–
STCF	BW–	STCF #4, r	C8 + zz + r : 24 : #4	r<#4> ← CY	−−−−−−	3	3.3.–
	BW–	STCF A , r	C8 + zz + r : 2C	r<A> ← CY	−−−−−−	2	3.3.–
	B––	STCF #3, (mem)	B0 + mem : A0 + #3	(mem)<#3> ← CY	−−−−−−	2 + M	7.–.–
	B––	STCF A , (mem)	B0 + mem : 2C	(mem)<A> ← CY	−−−−−−	2 + M	7.–.–
ANDCF	BW–	ANDCF #4, r	C8 + zz + r : 20 : #4	CY ← CY and r<#4>	−−−−−*	3	3.3.–
	BW–	ANDCF A , r	C8 + zz + r : 28	CY ← CY and r<A>	−−−−−*	2	3.3.–
	B––	ANDCF #3, (mem)	B0 + mem : 80 + #3	CY ← CY and (mem)<#3>	−−−−−*	2 + M	6.–.–
	B––	ANDCF A , (mem)	B0 + mem : 28	CY ← CY and (mem)<A>	−−−−−*	2 + M	6.–.–
ORCF	BW–	ORCF #4, r	C8 + zz + r : 21 : #4	CY ← CY or r<#4>	−−−−−*	3	3.3.–
	BW–	ORCF A , r	C8 + zz + r : 29	CY ← CY or r<A>	−−−−−*	2	3.3.–
	B––	ORCF #3, (mem)	B0 + mem : 88 + #3	CY ← CY or (mem)<#3>	−−−−−*	2 + M	6.–.–
	B––	ORCF A , (mem)	B0 + mem : 29	CY ← CY or (mem)<A>	−−−−−*	2 + M	6.–.–
XORCF	BW–	XORCF #4, r	C8 + zz + r : 22 : #4	CY ← CY xor r<#4>	−−−−−*	3	3.3.–
	BW–	XORCF A , r	C8 + zz + r : 2A	CY ← CY xor r<A>	−−−−−*	2	3.3.–
	B––	XORCF #3, (mem)	B0 + mem : 90 + #3	CY ← CY xor (mem)<#3>	−−−−−*	2 + M	6.–.–
	B––	XORCF A , (mem)	B0 + mem : 2A	CY ← CY xor (mem)<A>	−−−−−*	2 + M	6.–.–
RCF	–––	RCF	10	CY ← 0	−−0−00	1	2
SCF	–––	SCF	11	CY ← 1	−−0−01	1	2
CCF	–––	CCF	12	CY ← not CY	−−X−0*	1	2
ZCF	–––	ZCF	13	CY ← not Z flag	−−X−0*	1	2
BIT	BW–	BIT #4, r	C8 + zz + r : 33 : #4	Z ← not r<#4>	X*1X0−	3	3.3.–
	B––	BIT #3, (mem)	B0 + mem : C8 + #3	Z ← not (mem)<#3>	X*1X0−	2 + M	6.–.–
RES	BW–	RES #4, r	C8 + zz + r : 30 : #4	r<#4> ← 0	−−−−−−	3	3.3.–
	B––	RES #3, (mem)	B0 + mem : B0 + #3	(mem)<#3> ← 0	−−−−−−	2 + M	7.–.–
SET	BW–	SET #4, r	C8 + zz + r : 31 : #4	r<#4> ← 1	−−−−−−	3	3.3.–
	B––	SET #3, (mem)	B0 + mem : B8 + #3	(mem)<#3> ← 1	−−−−−−	2 + M	7.–.–
CHG	BW–	CHG #4, r	C8 + zz + r : 32 : #4	r<#4> ← not r<#4>	−−−−−−	3	3.3.–
	B––	CHG #3, (mem)	B0 + mem : C0 + #3	(mem)<#3> ← not (mem)<#3>	−−−−−−	2 + M	7.–.–
TSET	BW–	TSET #4, r	C8 + zz + r : 34 : #4	Z ← not r<#4> : r<#4> ← 1	X*1X0−	3	4.4.–
	B––	TSET #3, (mem)	B0 + mem : A8 + #3	Z ← not (mem)<#3> (mem)<#3> ← 1	X*1X0−	2 + M	7.–.–
BS1	–W–	BS1F A, r	D8 + r : 0E	A ← 1 search r ; Forward	−−− −−	2	–.3.–
	–W–	BS1B A, r	D8 + r : 0F	A ← 1 search r ; Backward	−−− −−	2	–.3.–

Note: ①; 0 is set when the bit searched for is found, otherwise 1 is set and an undefined value is set in the A register.

- 900/L1 Instruction Lists (7/10)

(7) Special operations and CPU control

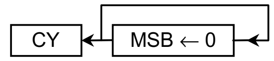
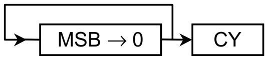
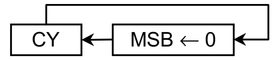
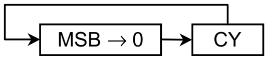
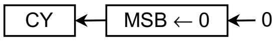
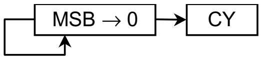
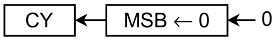
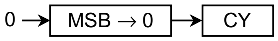
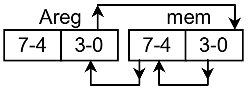
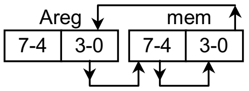
Group	Size	Mnemonic	Codes (16 hex)	Function	SZHVNC	Length (byte)	State
NOP	–––	NOP	00	no operation	––––––	1	2
EI	–––	EI [#3]	06 : #3	Sets interrupt enable flag. IFF ← #3	––––––	2	3
DI	–––	DI	06 : 07	Disables interrupt. IFF ← 7	––––––	2	4
PUSH	–W–	PUSH SR	02	(–XSP) ← SR	––––––	1	–.3.–
POP	–W–	POP SR	03	SR ← (XSP+)	* * * * * *	1	–.4.–
SWI	–––	SWI [#3]	F8 + #3	Software interrupt PUSH PC&SR JP (FFFF00H + 4 × #3)	––––––	1	19
HALT	–––	HALT	05	CPU halt	––––––	1	6
LDC	BWL BWL	LDC cr, r LDC r, cr	C8 + zz + r : 2E : cr C8 + zz + r : 2F : cr	cr ← r r ← cr	–––––– ––––––	3 3	3.3.3 3.3.3
LDX	B––	LDX (#8), #	F7:00 : #8 : 00 : # : 00	(#8) ← #	––––––	6	8.–.–
LINK	––L	LINK r, d16	E8 + r : 0C : d16	PUSH r LD r, XSP ADD XSP, d16	––––––	4	–.–.8
UNLK	––L	UNLK r	E8 + r : 0D	LD XSP, r POP r	––––––	2	–.–.7
LDF	–––	LDF #3	17 : #3	Seta register bank. RFP ← #3 (0 at reset)	––––––	2	2
INCF	–––	INCF	0C	Switches register banks. RFP ← RFP + 1	––––––	1	2
DECF	–––	DECF	0D	Switches register banks. RFP ← RFP – 1	––––––	1	2
SCC	BW–	SCC cc, r	C8 + zz + r : 70 + cc	if cc then r ← 1 else r ← 0	––––––	2	2.2.–

Note 1: When operand #3 coding in the EI instruction is omitted, 0 is used as the default value.

Note 2: When operand #3 coding in the SWI instruction is omitted, 7 is used as the default value.

- 900/L1 Instruction Lists (8/10)

(8) Rotate and Shift

Group	Size	Mnemonic		Codes (16 hex)		Function	SZHVNC	Length (byte)	State
RLC	BWL	RLC	#4, r	C8 + zz + r	: E8 : #4		**0P0*	3	3 + n/4
	BWL	RLC	A, r	C8 + zz + r	: F8		**0P0*	2	3 + n/4
	BW–	RLC<W>	(mem)	80 + zz + mem	: 78		**0P0*	2 + M	6.6
RRC	BWL	RRC	#4, r	C8 + zz + r	: E9 : #4		**0P0*	3	3 + n/4
	BWL	RRC	A, r	C8 + zz + r	: F9		**0P0*	2	3 + n/4
	BW–	RRC<W>	(mem)	80 + zz + mem	: 79		**0P0*	2 + M	6.6
RL	BWL	RL	#4, r	C8 + zz + r	: EA : #4		**0P0*	3	3 + n/4
	BWL	RL	A, r	C8 + zz + r	: FA		**0P0*	2	3 + n/4
	BW–	RL<W>	(mem)	80 + zz + mem	: 7A		**0P0*	2 + M	6.6
RR	BWL	RR	#4, r	C8 + zz + r	: EB : #4		**0P0*	3	3 + n/4
	BWL	RR	A, r	C8 + zz + r	: FB		**0P0*	2	3 + n/4
	BW–	RR<W>	(mem)	80 + zz + mem	: 7B		**0P0*	2 + M	6.6
SLA	BWL	SLA	#4, r	C8 + zz + r	: EC : #4		**0P0*	3	3 + n/4
	BWL	SLA	A, r	C8 + zz + r	: FC		**0P0*	2	3 + n/4
	BW–	SLA<W>	(mem)	80 + zz + mem	: 7C		**0P0*	2 + M	6.6
SRA	BWL	SRA	#4, r	C8 + zz + r	: ED : #4		**0P0*	3	3 + n/4
	BWL	SRA	A, r	C8 + zz + r	: FD		**0P0*	2	3 + n/4
	BW–	SRA<W>	(mem)	80 + zz + mem	: 7D		**0P0*	2 + M	6.6
SLL	BWL	SLL	#4, r	C8 + zz + r	: EE : #4		**0P0*	3	3 + n/4
	BWL	SLL	A, r	C8 + zz + r	: FE		**0P0*	2	3 + n/4
	BW–	SLL<W>	(mem)	80 + zz + mem	: 7E		**0P0*	2 + M	6.6
SRL	BWL	SRL	#4, r	C8 + zz + r	: EF : #4		**0P0*	3	3 + n/4
	BWL	SRL	A, r	C8 + zz + r	: FF		**0P0*	2	3 + n/4
	BW–	SRL<W>	(mem)	80 + zz + mem	: 7F		**0P0*	2 + M	6.6
RLD	B––	RLD	[A,](mem)	80 + mem	: 06		**0P0–	2 + M	14.–.–
RRD	B––	RRD	[A,](mem)	80 + mem	: 07		**0P0–	2 + M	14.–.–

Note 1: When #4/A is used to specify the number of shifts, module 16 (0 to 15) is used. Code 0 means 16 shifts.

Note 2: When the following instructions are used in the TLCS-90, the S, Z and V flags do not change.

RLCA, RRCA, RLA, RRA, SLAA, SRAA, SLLA, and SRLA
In the TLCS-900, these flags change.

- 900/L1 Instruction Lists (9/10)

(9) Jump, Call and Return

Group	Size	Mnemonic	Codes (16 hex)	Function	SZHVNC	Length (byte)	State
JP	–––	JP #16	1A : #16	PC ← #16	––––––	3	5
	–––	JP #24	1B : #24	PC ← #24	––––––	4	6
	–––	JR [cc,]$ + 2 + d8	60 + cc : d8	if cc then PC ← PC + d8	––––––	2	5/2 (T/F)
	–––	JRL [cc,]$ + 3 + d16	70 + cc : d16	if cc then PC ← PC + d16	––––––	3	5/2 (T/F)
	–––	JP [cc,]mem	B0 + mem : D0 + cc	if cc then PC ← mem	––––––	2 + M	7/4 (T/F)
CALL	–––	CALL #16	1C : #16	PUSH PC : JP #16	––––––	3	9
	–––	CALL #24	1D : #24	PUSH PC : JP #24	––––––	4	10
	–––	CALR $ + 3 + d16	1E : d16	PUSH PC : JR $ + 3 + d16	––––––	3	10
	–––	CALL [cc,]mem	B0 + mem : E0 + cc	if cc then PUSH PC : JP mem	––––––	2 + M	12/4 (T/F)
DJNZ	BW–	DJNZ [r,]$ + 3/4 + d8	C8 + zz + r : 1C : d8	r ← r – 1 if r ≠ 0 then JR $ + 3 + d8	––––––	3	6 (r ≠ 0) 4 (r = 0)
RET	–––	RET	0E	POP PC	––––––	1	9
	–––	RET cc	B0 : F0 + cc	if cc then POP PC	––––––	2	12/4 (T/F)
	–––	RETD d16	0F : d16	RET : ADD XSP, d16	––––––	3	11
	–––	RETI	07	POP SR&PC	******	1	12

Note 1: (T/F) represents the number of states at true / false.

- Instruction Lists of 900/L1 (10/10)

(10) Addressing mode

Classification	mode	state
R	R	+0
r	r	+1
(mem)	(R)	+0
	(R + d8)	+1
	(#8)	+1
	(#16)	+2
	(#24)	+3
	(r)	+1
	(r + d16)	+3
	(r + r8)	+3
	(r + r16)	+3
	(−r)	+1
	(r+)	+1

(11) Interrupt

mode		operation	state
General-purpose interrupt processing		PUSH PC PUSH SR IFF ← accepted level + 1 INTNEST ← INTNEST + 1 JP (FFFF00H + vector)	18
micro DMA	I/O to MEM	(DMADn+) ← (DMASn)	8. 8. 12
	I/O to MEM	(DMADn−) ← (DMASn)	8. 8. 12
	MEM to I/O	(DMADn) ← (DMASn+)	8. 8. 12
	MEM to I/O	(DMADn) ← (DMASn−)	8. 8. 12
	I/O to I/O	(DMADn) ← (DMASn)	8. 8. 12
	Counter	DMASn ← DMASn + 1	−. −. 5

Note: For details of interrupt processing, refer to the “Interrupts” section.

Appendix C Instruction Code Maps (1/4)

1-byte op code instructions

H/L	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	☒	PUSH SR	POP SR	☒	HALT	EI n	RETI	LD (n) , n	PUSH n	LDW (n) , nn	PUSHW nn	INCF	DECF	RET	RETD dd
1	RCF	SCF	CCF	ZCF	PUSH A	POP A	EX F, F'	LDF n	PUSH F	POP F	JP nn	JP nnn	CALL nn	CALL nnn	CALR PC＋dd	
2	LD R, n								PUSH RR							
3	LD RR, nn								PUSH XRR							
4	LD XRR, nnnn								POP RR							
5									POP XRR							
6	F	LT	LE	ULE	PE/OV	M/MI	Z	JR C	cc,PC＋d (T)	GE	GT	UGT	PO/NOV	P/PL	NZ	NC
7	F	LT	LE	ULE	PE/OV	M/MI	Z	JRL C	cc,PC＋dd (T)	GE	GT	UGT	PO/NOV	P/PL	NZ	NC
8	scr. B (XWA)	(XBC)	(XDE)	(XHL)	(XIX)	(XIY)	(XIZ)	(XSP)	scr. B (XWA +d)	(XBC +d)	(XDE +d)	(XHL +d)	(XIX +d)	(XIY +d)	(XIZ +d)	(XSP +d)
9	scr. W (XWA)	(XBC)	(XDE)	(XHL)	(XIX)	(XIY)	(XIZ)	(XSP)	scr. W (XWA +d)	(XBC +d)	(XDE +d)	(XHL +d)	(XIX +d)	(XIY +d)	(XIZ +d)	(XSP +d)
A	scr. L (XWA)	(XBC)	(XDE)	(XHL)	(XIX)	(XIY)	(XIZ)	(XSP)	scr. L (XWA +d)	(XBC +d)	(XDE +d)	(XHL +d)	(XIX +d)	(XIY +d)	(XIZ +d)	(XSP +d)
B	dst (XWA)	(XBC)	(XDE)	(XHL)	(XIX)	(XIY)	(XIZ)	(XSP)	dst (XWA +d)	(XBC +d)	(XDE +d)	(XHL +d)	(XIX +d)	(XIY +d)	(XIZ +d)	(XSP +d)
C	scr. B (n)	(nn)	(nnn)	(mem)	(−xrr)	(xrr+)		reg. B r	reg. B W	A	B	C	D	E	H	L
D	scr. W (n)	(nn)	(nnn)	(mem)	(−xrr)	(xrr+)		reg. W rr	reg. W WA	BC	DE	HL	IX	IY	IZ	SP
E	scr. L (n)	(nn)	(nnn)	(mem)	(−xrr)	(xrr+)		reg. L xrr	reg. L XWA	XBC	XDE	XHL	XIX	XIY	XIZ	XSP
F	dst (n)	(nn)	(nnn)	(mem)	(−xrr)	(xrr+)		LDX (n), n	SWI n 0	1	2	3	4	5	6	7

Note 1: Codes in blank parts are undefined instructions (i.e., illegal instructions).

Note 2: Dummy instructions are assigned to codes 01H and 04H. Do not use them.

Appendix C Instruction Code Maps (2/4)

1st byte: reg

H/L	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				LD r,#	PUSH r	POP r	CPL BW r	NEG BW r	MUL rr,#	MULS rr,#	DIV rr,#	DIVS BW rr,#	LINK L r, dd	UNLK L r	BS1F A, r	BS1B W A, r
1	DAA B r		EXTZ WL r	EXTS WL r	PAA WL r		MIRR W r			MULA W r	╳	╳	DJNZ BW r, d			
2	ANDCF #, r	ORCF #, r	XORCF #, r	LDCF #, r	STCF BW #, r				ANDCF A, r	ORCF A, r	XORCF A, r	LDCF A, r	STCF BW A, r		LDC cr, r	LDC r, cr
3	RES #, r	SET #, r	CHG #, r	BIT #, r	TSET BW #, r				MINC1	MINC2 #, r	MINC4 W	╳	MDEC1	MDEC2 #, r	MDEC4 W	╳
4				MUL	R, r			BW				MULS	R, r			BW
5				DIV	R, r			BW				DIVS	R, r			BW
6	8	1	2	INC 3	#3, r 4	5	6	7	8	1	2	DEC 3	#3, r 4	5	6	7
7	F	LT	LE	ULE	PE/OV	M/MI	Z	SCC C	cc, r (T)	GE	GT	UGT	PO/NOV	P/PL	NZ	BW NC
8				ADD	R, r							LD	R, r			
9				ADC	R, r							LD	r, R			
A				SUB	R, r				0	1	2	LD 3	r, #3 4	5	6	7
B				SBC	R, r							EX	R, r			BW
C				AND	R, r				ADD r, #	ADC r, #	SUB r, #	SBC r, #	AND r, #	XOR r, #	OR r, #	CP r, #
D				XOR	R, r				0	1	2	CP 3	r, #3 4	5	6	BW 7
E				OR	R, r				RLC #, r	RRC #, r	RL #, r	RR #, r	SLA #, r	SRA #, r	SLL #, r	SRL #, r
F				CP	R, r				RLC A, r	RRC A, r	RL A, r	RR A, r	SLA A, r	SRA A, r	SLL A, r	SRL A, r

r: Register specified by the 1st byte code. (Any CPU registers can be specified.)

R: Register specified by the 2nd byte code. (Only eight current registers can be specified.)

B: Operand size is a byte.

W: Operand size is a word.

L: Operand size is a long word.

Note: Dummy instructions are assigned to codes 1AH, 1BH, 3BH, and 3FH. Do not use them.

Appendix C Instruction Code Maps (3/4)

1st byte: src (mem)

H/L	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0					PUSH BW (mem)		RLD	RLD B A, (mem)								
1	LDI	LDIR	LDD	LDDR BW	CPI	CPIR	CPD	CPDR BW		LD BW (nn), (m)						
2				LD	R,(mem)											
3				EX	(mem),R			BW	ADD	ADC	SUB	SBC (mem), #	AND	XOR	OR	CP BW
4				MUL	R,(mem)			BW				MULS	R,(mem)			BW
5				DIV	R,(mem)			BW				DIVS	R,(mem)			BW
6	8	1	2	INC 3	#3, (mem) 4	5	6	BW 7	8	1	2	DEC 3	#3, (mem) 4	5	6	BW 7
7									RLC	RRC	RL	RR (mem)	SLA	SRA	SLL	SRL BW
8				ADD	R,(mem)							ADD	(mem),R			
9				ADC	R,(mem)							ADC	(mem),R			
A				SUB	R,(mem)							SUB	(mem),R			
B				SBC	R,(mem)							SBC	(mem),R			
C				AND	R,(mem)							AND	(mem),R			
D				XOR	R,(mem)							XOR	(mem),R			
E				OR	R,(mem)							OR	(mem),R			
F				CP	R,(mem)							CP	(mem),R			

B: Operand size is a byte.

W: Operand size is a word.

Appendix C Instruction Code Maps (4/4)

1st byte: dst (mem)

H/L	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	LD B (m), #		LD W (m), #		POP B (mem)		POP W (mem)									
1					LD B (m), (nn)		LD W (m), (nn)									
2				LDA	R,(mem)			W	ANDCF	ORCF	XORCF A, (mem)	LDCF	STCF B			
3				LDA	R,(mem)			L								
4				LD	(mem),R			B								
5				LD	(mem),R			W								
6				LD	(mem),R			L								
7																
8	0	1	2	ANDCF 3	#3, (mem) 4	5	6	B 7	0	1	2	ORCF 3	#3, (mem) 4	5	6	B 7
9	0	1	2	XORCF 3	#3, (mem) 4	5	6	B 7	0	1	2	LDCF 3	#3, (mem) 4	5	6	B 7
A	0	1	2	STCF 3	#3, (mem) 4	5	6	B 7	0	1	2	TSET 3	#3, (mem) 4	5	6	B 7
B	0	1	2	RES 3	#3, (mem) 4	5	6	B 7	0	1	2	SET 3	#3, (mem) 4	5	6	B 7
C	0	1	2	CHG 3	#3, (mem) 4	5	6	B 7	0	1	2	BIT 3	#3, (mem) 4	5	6	B 7
D	F	LT	LE	ULE	PE/OV	M/MI	Z	JP C	cc, mem (T)	GE	GT	UGT	PO/NOV	P/PL	NZ	NC
E	F	LT	LE	ULE	PE/OV	M/MI	Z	CALL C	cc, mem (T)	GE	GT	UGT	PO/NOV	P/PL	NZ	NC
F	F	LT	LE	ULE	PE/OV	M/MI	Z	RET C	cc (T)	(1st byte code is B0H.) GE	GT	UGT	PO/NOV	P/PL	NZ	NC

B: Operand size is a byte.

W: Operand size is a word.

L: Operand size is a long word.

Appendix D Differences between TLCS-90 and TLCS-900/L1 Series

Series Item	TLCS-90	TLCS-900/L1
CPU architecture	8-bit CPU	16-bit CPU
Built-in ROM/built-in RAM	8-bit data bus	16-bit data bus
Built-in I/O	8-bit data bus	8-bit data bus
External data bus	8-bit data bus	8-bit/16-bit data bus (can be mixed)
Program space (except devices with MMU)	64 KB	16 MB (linear)
Data space	16 MB (bank)	16 MB (linear)
Instruction set/instruction mnemonic	TLCS-90	TLCS-90 + α α = enhancement of 16-bit multiply / divide instructions and bit operation instruction, 32-bit load/operation instructions, C compiler instructions, register bank operation instructions, etc.
Instruction code (object code)	Unique to TLCS-90	Unique to TLCS-900 (Different from TLCS-90.)
Addressing mode	TLCS-90	TLCS-90 + α α = (-Reg), (Reg+), (Reg + disp16), (Reg + Reg16), (nnn)
General-purpose register	TLCS-90	TLCS-90 + α α = Uses as 32 bits and register bank, and adds a system stack pointer.
Flag (F)	S Z I H X V N C	S Z 0 H 0 V N C I flag is extended to IFF2 to 0 of status register.X flag is deleted.

Reset	PC ← 0000H
	(SP does not change.)
		PC ← (Vector base address)
		XSP ← 100H

Built-in ROM address	0000H to	Undefined
Built-in RAM address	to FFxxH	Undefined
Built-in I/O address	FFxxH to FFFFH	000000H to 000FFFH
Direct addressing area (n)	FF00H to FFFFH	000000H to 0000FFH
Interrupt		
Interrupt start address	0000H + (8 × V)	Vector base address + 4 × V
Register to be saved	PC & AF	PC & SR
Mask register	IFF	IFF2 to 0
Mask level	0, 1	0 to 7

Series Item	TLCS-90	TLCS-900/L1
Instruction		
1. ADD R, r (word type)	S/Z/V flags don’t change. [S/Z/V flag changes expect add 16 bit register.]	S/Z/V flag changes.
2. Shift of A register	「 RLCA RRCA RLA RRA SLAA SRAA SLLA SRLA 」 S/Z/V flags don’t change in these instruction. 「 RLC A RRC A RL A RR A SLA A SRA A SLL A SRL A 」 S/Z/V flag changes in these instruction.	S/Z/V flag changes.

Note: The TLCS-900/L1 series is essentially the same as the TLCS-90 series but with a 16-bit CPU. Built-in I/Os are completely compatible with those of the TLCS-90.

However, six types of instructions used in the TLCS-90 series do not directly correspond with those used in the TLCS-900/L1 series. Thus, when transfering programs designed for the TLCS-90 to the TLCS-900/L1, replace them with equivalents as follows:

Instructions in TLCS-90 but not in TLCS-900/L1	Equivalent instructions in TLCS-900/L1
EXX	EX BC, BC' EX DE, DE' EX HL, HL'
EX AF, AF'	EX A, A' EX F, F'
PUSH AF	PUSH A PUSH F
POP AF	POP F POP A
INCX	(32-bit INC instruction)
DECX	(32-bit DEC instruction)

Some TLCS-900/L1 series instructions, though basically the same as TLCS-90 instructions, have more functions and more specification items in their operands. They are listed below.

TLCS-90	TLCS-900/L1
INC reg INC mem	INC imm3, reg INC imm3, mem
DEC reg DEC mem	DEC imm3, reg DEC imm3, mem
RLC reg RRC reg RL reg RR reg SLA reg SRA reg SLL reg SRL reg	RLC imm, reg RRC imm, reg RL imm, reg RR imm, reg SLA imm, reg SRA imm, reg SLL imm, reg SRL imm, reg