MF297-06a

EPSON

CMOS 4-BIT SINGLE CHIP MICROCOMPUTER

E0C6200/6200A CORE CPU MANUAL

ENERGY SAVING EPSON

SEIKO EPSON CORPORATION

E0C6200/6200A Core CPU Manual

CONTENTS

1 DESCRIPTION

The E0C6200/6200A is the Core CPU of the E0C62 Family of CMOS 4-bit single-chip microcomputers. The CPU features a highly-integrated architecture. Memory-mapped peripheral circuits can include RAM, ROM, I/O ports, interrupt controllers, timers and LCD drivers, depending upon the application.

The memory address space is divided into program and data memory, each with data and address lines. Program memory consists of on-chip ROM, containing instructions to be executed by the CPU. Data memory consists of RAM and memory-mapped I/O, as determined by the design of the peripheral circuitry.

A large memory as well as instructions capable of 8-bit data manipulation enhance the functionality of the E0C62 Family. Implementation of a common Core CPU ensures that a wide range of application-specific devices can be designed and fabricated with the minimum turnaround time.

1.1 System Features

- Common Core CPU for all E0C62 Family microcomputers
- UP to 8,192 12-bit words of program memory (ROM)
- UP to 4,096 4-bit words of data memory (RAM/peripheral circuits)
- Memory-mapped I/O
- 5, 7 or 12 clock cycle instructions
- 109 instructions
- Up to 85 levels of subroutine nesting
- 8-bit stack pointer
- Up to 15 interrupt vectors
- Two standby modes
- Low-power CMOS process

1.2 Instruction Set Features

- Four addressing modes: one direct, two indirect, and one stack pointer
- Direct addressing transfers data to and from data memory with a single instruction, resulting in more efficient code
- 8-bit load instructions and table look-up instructions
- Arithmetic operations in either hexadecimal or decimal
- Arithmetic and logical instructions: addition, subtraction, logical AND, OR, exclusive-OR, comparison and rotation

1.3 Differences between E0C6200 and E0C6200A

There are some differences in the following operation/circuit between the E0C6200 and the E0C6200A. For the detailes of each difference, refer to the section enclosed with parentheses.

- Initial setting of D (decimal) flag (refer to Section 2.5.5, "Initial reset".)
- Interrupt circuit
  - Interrupt timing (refer to Section 2.5.3, "Operation during interrupt generation".)
  - Writing to interrupt mask registers and reading of interrupt flags (refer to Appendix A, "E0C6200A (Advanced E0C6200) Core CPU".)

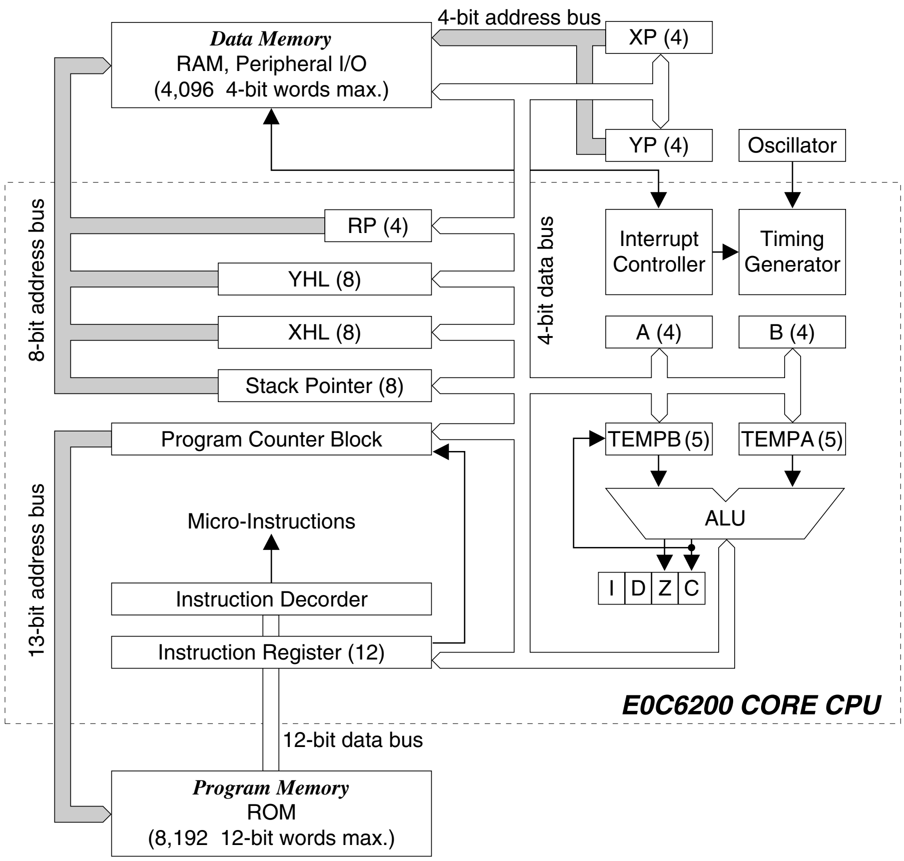


Fig. 1.1 Block diagram

2 MEMORY AND OPERATIONS

A single-chip microcomputer using the E0C6200/6200A Core CPU has four major blocks: the program memory (ROM), the data memory (RAM and I/O), the arithmetic logic unit (ALU) and the timing generator circuit. This section describes each of these blocks in detail.

2.1 Program Memory (ROM)

Program memory contains the instructions that the CPU executes. Figure 2.1.1 shows the configuration of the program memory.

Each instruction is a 12-bit word. Program memory can also be used for data tables for the table look-up instructions.

There are two banks of program memory. Each bank is subdivided into 16 pages of 256 words (or steps). That is:

Program memory	= 2 banks
	= 8,192 steps
1 bank	= 4,096 steps
	= 16 pages
1 page	= 256 steps
1 step	= 1 word
	= 12 bits

Certain addresses in ROM have specific functions, as shown in Table 2.1.1.

Table 2.1.1 Allocated program memory

Address	Function
Bank 0, Page 1, Step 0	Reset vector
Bank 0, Page 1, Step 1 to 15	Interrupt vectors used while a program is running in bank 0
Bank 0, Page 0, Step 0 to 255	Bank 0, page 0 area Direct call subroutines for use by CALZ while a program is running in bank 0
Bank 1, Page 1, Step 1 to 15	Interrupt vectors used while a program is running in bank 1
Bank 1, Page 0, Step 0 to 255	Bank 1, page 0 area Direct call subroutines for use by CALZ while a program is running in bank 1

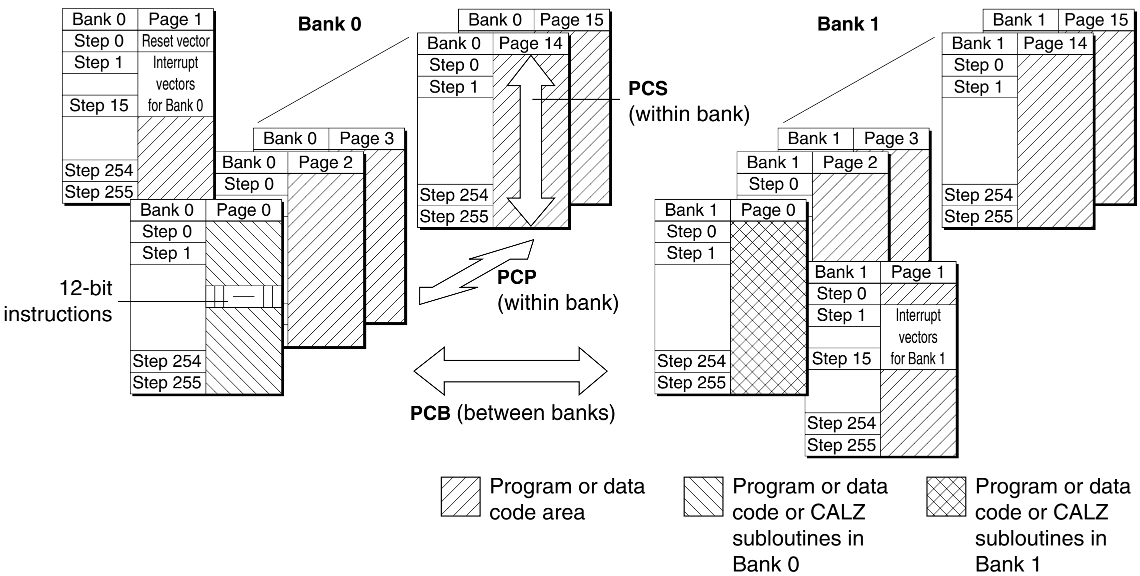


Fig. 2.1.1 Program memory configuration

2.1.1 Program counter block

The program counter is used to point to the next instruction step to be executed by the CPU. See Figure 2.1.1.1.

The program counter has the following registers.

Table 2.1.1.1 Program counter registers

Register	Size
PCB (Program Counter-Bank)	1-bit register
PCP (Program Counter-Page)	4-bit counter
PCS (Program Counter-Step)	8-bit counter
NBP (New Bank Pointer)	1-bit register
NPP (New Page Pointer)	4-bit register

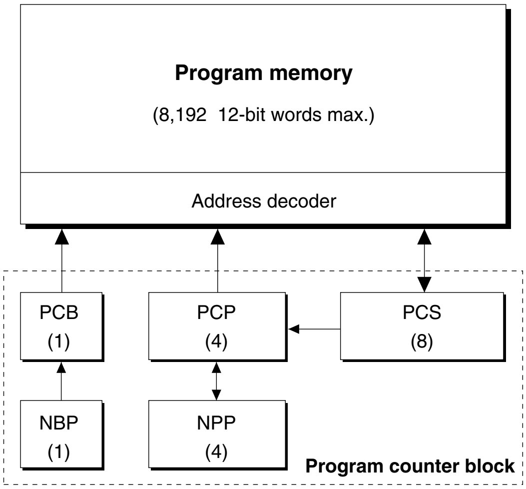


Fig. 2.1.1.1 Program counter configuration

PCB, PCP and PCS together from a 13-bit counter which can address any location in program memory.

PCP and PCS together from a 12-bit counter which can address any location within a given bank of program memory. Each time an instruction other than a jump is executed, this counter increments by one. Thus, a jump instruction does not need to be executed between the last step of one page and the first step of the next.

The contents of NBP and NPP are loaded into PCB and PCP each time an instruction is executed. On reset, NBP and NPP are loaded with the same values as PCB and PCP.

2.1.2 Flags

The following flags are provided.

Table 2.1.2.1 Flags

Flag	Menus	Size
Interrupt	I	1: Enabled 0: Disabled
Decimal mode	D	1: Decimal 0: Hexadecimal
Zero	Z	1: Set 0: Ignored
Carry	C	1: Set 0: Ignored

2.1.3 Jump instructions

A jump can be made using the instructions in Table 2.1.3.1.

Table 2.1.3.1 Jump instructions

Type of jump	Instruction
Unconditional	JP
Conditional	JP C, JP NC, JP Z, JP NZ
Subroutine call	CALL, CALZ
Return	RET, RETS, RETD
Page set	PSET
Indirect	JPBA

The differences between jumps within the same page and jumps from one page to another is as follows.

- Jumps within the same page
  A jump can be made within the same page using any of the following instructions:

  JP, JP C, JP Z, JP NZ, JPBA or CALL

  The destination address is specified by the 8-bit operand. A label can be used to specify a destination address with the E0C62 Family cross assembler.
- Jumps from one page to another
  The destination bank and page should be set using PSET before executing a JP instruction.

2.1.4 PSET with jump instructions

PSET loads the four low-order bits (page part) of its 5-bit operand to NPP (new page pointer) and loads the high-order bit (bank part) to NBP (new bank pointer). Executing a JP instruction immediately after PSET causes a jump to the bank specified by NBP, the page specified by NPP and the step specified by the JP instruction operand. See Figure 2.1.4.1.

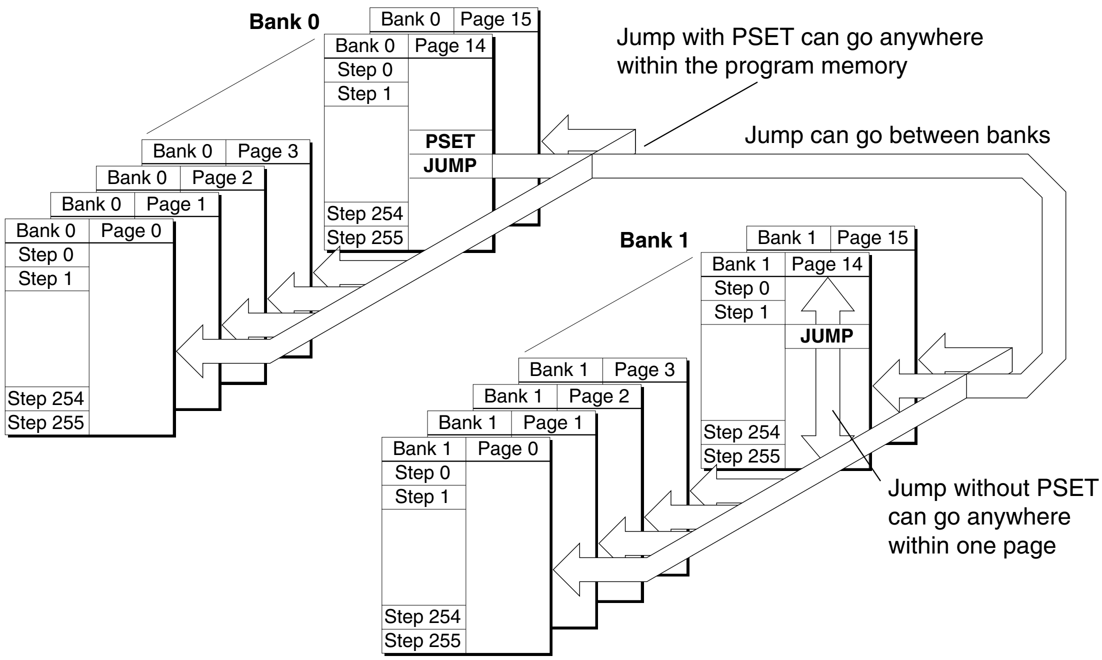


Fig. 2.1.4.1 The PSET and jump instructions

2.1.5 Call instructions

As only the page data specified by NPP is loaded to PCP when a call instruction is executed, subroutine calls between banks are not possible. Jumps between banks can only be made using JP instructions.

2.1.6 PSET instruction

Jump or call instructions must follow PSET immediately in order for PSET to affect the destination address. When a jump or call is not immediately preceded by PSET, the destination address is within the current page.

Some examples using PSET are shown in Table 2.1.6.1.

Table 2.1.6.1 PSET examples

Bank	Page	Stap	Instruction	Operation
0	01H	10H	PSET 13H	
0	01H	11H	JP 08H	The program jumps to bank 1, page 3, step 8.
•	•	•	•	
•	•	•	•	
0	01H	21H	PSET 15H	
0	01H	22H	NOP5	The data set by PSET is canceled.
0	01H	23H	JP 09H	The program jumps to bank 0, page 1, step 9.
•	•	•	•	
•	•	•	•	
0	01H	55H	SCF	C flag is set.
0	01H	56H	PSET 14H	
0	01H	57H	JP C, 07H	The program jumps to bank 1, page 4, step 7 because C flag = 1.
•	•	•	•	
•	•	•	•	
0	01H	60H	RFC	C flag is reset.
0	01H	61H	PSET 05H	
0	01H	62H	JP C, 08H	No jump occurs because C flag = 0.
0	01H	63H	JP 09H	The data set by PSET is canceled, and the program jumps to bank 0, page 1, step 9.
•	•	•	•	
•	•	•	•	

2.1.7 CALZ instruction

CALZ is a direct subroutine call instruction. It calls a subroutine, in page 0 of the current bank, from any page without requiring the use of PSET.

If CALZ is executed immediately after PSET, the bank and page set by PSET is canceled. This allows direct subroutine calls to page 0, minimizing repeated code and unnecessary use of PSET. See Figure 2.1.7.1.

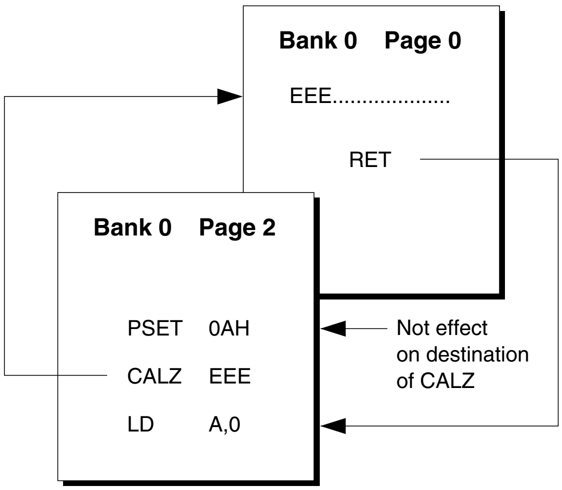


Fig. 2.1.7.1 The use of the CALZ instruction

The difference between CALL and CALZ is shown in Figure 2.1.7.2.

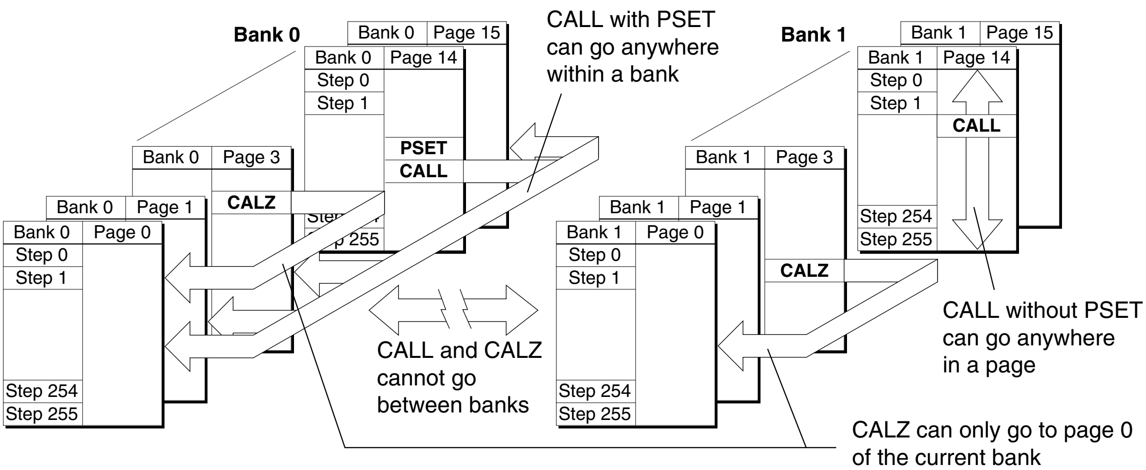


Fig. 2.1.7.2 The difference between CALL and CALZ instructions

2.1.8 RET and RETS instructions

The RET instruction causes a return from a subroutine to the address immediately following the address from where that subroutine was called. The RETS instruction causes a return to the address following this address. Proper use of RET and RETS allows simple conditional exits subroutines back to the main routine. See Figure 2.1.8.1.

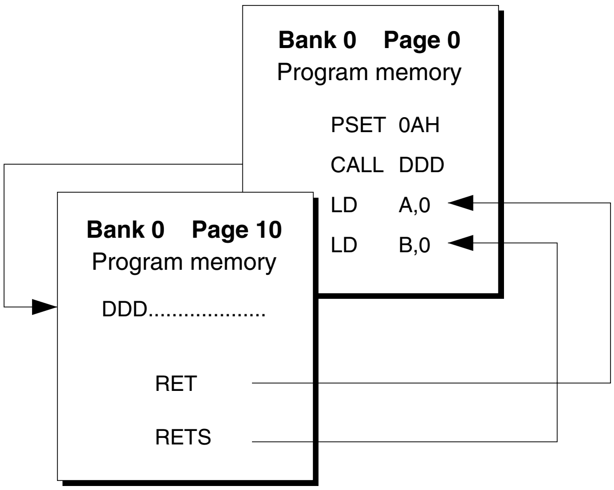


Fig. 2.1.8.1 Difference between RET and RETS instructions

2.1.9 Stack considerations for call instructions

When a subroutine is called, the return address is loaded into the stack and retrieved when control is returned to the calling program. Nesting allows efficient usage of the stack area.

As the stack area resides in the data memory, care should be taken to ensure that the stack area is not corrupted by other data.

2.2 Data Memory

The data memory area comprises 4,096 4-bit words. The RAM, timer, I/O and other peripheral circuits are mapped into this memory according to the designer's specifications. Figure 2.2.1 shows the data memory configuration.

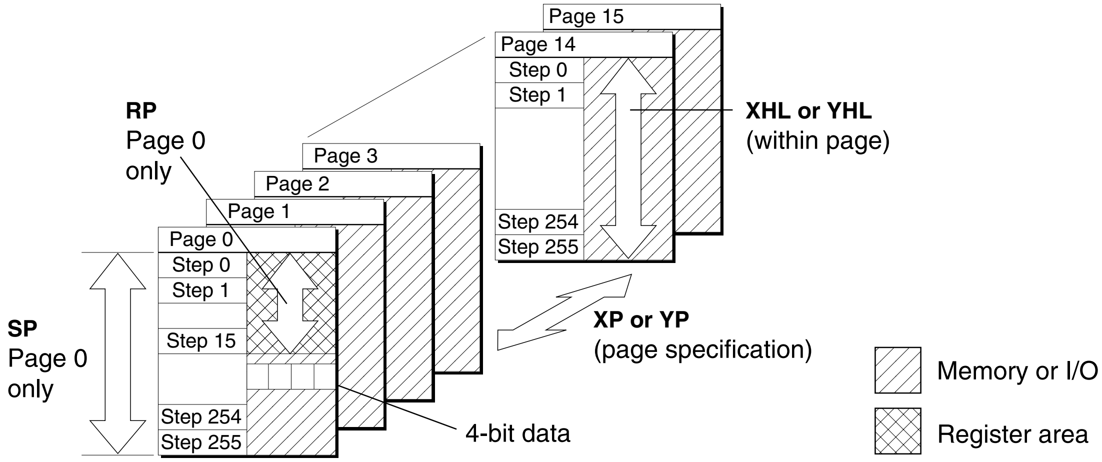


Fig. 2.2.1 Data memory configuration

2.2.1 Data memory addressing

The following registers and pointers, which are described in detail below, are used to address the data memory.

Table 2.2.1.1 Registers and pointer for data memory addressing

Register/Pointer	Mnemonic	Size (bits)
Index Register X	IX	12
Index Register Y	IY	12
Stack Pointer	SP	8
Register	RP	4

- **Index register IX**

  Index register IX has a 4-bit page part (XP) and an 8-bit register (XHL), and can address any location in the data memory. See Figure 2.2.1.1.

  XHL is divided into two 4-bit groups: the four high-order bits (XH) and the four low-order bits (XL), and can address any location within a page.

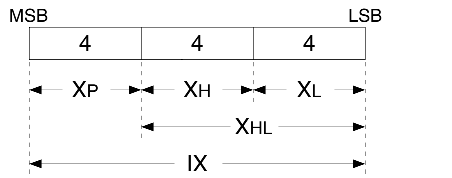


Fig. 2.2.1.1 The configuration of the index register IX

– MX is the data memory location whose address is specified by IX.

– M(X) refers to the contents of the data memory location whose address is specified by IX.

– XHL can be incremented by 1 or 2 using a post-increment instruction (LDPX, ACPX, SCPX, LBPX or RETD). An overflow occurring in XHL does not affect the flags.

- **Index register IY**

Index register IY is like the index register IX: it has a 4-bit page part (YP), an 8-bit register (YHL), and can address any location in the data memory. See Figure 2.2.1.2.

YHL is divided into two 4-bit groups: the four high-order bits (YH) and the four low-order bits (YL), and can address any location within a page.

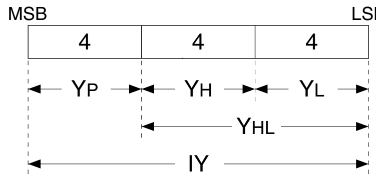


Fig. 2.2.1.2 The configuration of the index register IY

– MY is the data memory location whose address is specified by IY.

– M(Y) refers to the contents of the data memory location whose address is specified by IY.

– YHL can be incremented by 1 using a post-increment instruction (LDPY, ACPY or SCPY). An overflow occurring in YHL does not affect the flags.

- **Stack pointer SP**

The stack area resides in the data memory. The 8-bit, push-down/pop-up stack pointer (SP) is used to address an element within the stack.

Since it is an 8-bit pointer, SP can only address 256 words out of the total 4,096 words of data memory. When SP is used, the high-order 4 bits (page part) of the data memory address are 0, giving a stack area of 256 words in the address range 000H to 0FFH.

In systems with a RAM area of less than 256 words, the entire RAM area can be used as the stack area.

Stack area usage is shown in Table 2.2.1.2.

Table 2.2.1.2 Stack usage

Operation	Instruction	Stack usage
Push-down	Interrupt	-3
(SP is decremented)	CALL or CALZ	-3
	PUSH	-1
	DEC SP	-1
Pop-up	RET, RETS or RETD	+3
(SP is incremented)	POP	+1
	INC SP	+1

The PUSH instruction can be used to store registers and flags in the stack in single-word (4-bit) units. The POP instruction is used to retrieve this data.

When an interrupt occurs or a call instruction is executed, the return address from the program counter is pushed onto the stack. When a return instruction is executed, the return address is retrieved from the stack and loaded into the program counter.

On an interrupt, only the program counter is saved on the stack; flag and register data are not saved. Programs should be designed so that flag and register data are pushed onto the stack by the interrupt service routines.

Following a system reset, SP should be initialized using the LD SPH,*r* or LD SPL,*r* instructions, where *r* represents A, B, MX or MY (4 bits).

Stack pointer data can be read using LD *r*,SPH or LD *r*,SPL.

- **Register pointer RP**

The register pointer (RP) is a 4-bit register used to address the first 16 words of data memory, or the register area. Direct addressing can be used to read from, write to, increment or decrement any location within this area efficiently, using a single instruction.

Programs cannot directly access RP. It uses the operand of direct addressing instructions. The instructions that can access the register area of data memory are:

LD A,M*n*	A ← M(*n*)
LD B,M*n*	B ← M(*n*)
LD M*n*,A	M(*n*) ← A
LD M*n*,B	M(*n*) ← B
INC M*n*	M(*n*) ← M(*n*) + 1
DEC M*n*	M(*n*) ← M(*n*) – 1
n: 0 to F	

where M(*n*) is the contents of a data memory location within the register area.

As the register area can also be indirectly accessed using IX, IY or SP, the stack area should not grow to address 000H to 00FH when RP is used.

2.3 ALU (Arithmetic Logic Unit) and Registers

Table 2.3.1 shows ALU operations between the 4-bit registers, TEMPA and TEMPB.

Table 2.3.1 ALU register operation

Operation	Instruction
Add, without carry	ADD
Add, with carry	ADC
Subtract, without borrow	SUB
Subtract, with borrow	SBC
Logical-AND	AND
Logical-OR	OR
Exclusive-OR	XOR
Comparison	CP
Flag bit test	FAN
Rotate right, with carry	RRC
Rotate left, with carry	RLC
Invert	NOT

The Z (zero) flag is set when the result of ALU operation is

C	3	2	1	0
X	0	0	0	0

X: Don't care.

The C (carry) flag is set when an add operation causes a carry or when a subtract operation causes a borrow.

2.3.1 D (decimal) flag and decimal operations

Setting the D (decimal) flag activates the decimal mode, allowing decimal addition and subtraction. Table 2.3.1.1 shows the relations of actual (decimal) results, ALU outputs, and the values of the C and Z flags.

Table 2.3.1.1 Results of hexadecimal and decimal operations

Addition							Subtraction						
Actual result	D = 0 : Result of hexadecimal operation			D = 1 : Result of decimal operation			Actual result	D = 0 : Result of hexadecimal operation			D = 1 : Result of decimal operation		
	Z	C	ALU output	Z	C	ALU output		Z	C	ALU output	Z	C	ALU output
0	1	0	0	1	0	0	-16	1	1	0	0	1	A
1	0	0	1	0	0	1	-15	0	1	1	0	1	B
2	0	0	2	0	0	2	-14	0	1	2	0	1	C
3	0	0	3	0	0	3	-13	0	1	3	0	1	D
4	0	0	4	0	0	4	-12	0	1	4	0	1	E
5	0	0	5	0	0	5	-11	0	1	5	0	1	F
6	0	0	6	0	0	6	-10	0	1	6	1	1	0
7	0	0	7	0	0	7	-9	0	1	7	0	1	1
8	0	0	8	0	0	8	-8	0	1	8	0	1	2
9	0	0	9	0	0	9	-7	0	1	9	0	1	3
10	0	0	A	1	1	0	-6	0	1	A	0	1	4
11	0	0	B	0	1	1	-5	0	1	B	0	1	5
12	0	0	C	0	1	2	-4	0	1	C	0	1	6
13	0	0	D	0	1	3	-3	0	1	D	0	1	7
14	0	0	E	0	1	4	-2	0	1	E	0	1	8
15	0	0	F	0	1	5	-1	0	1	F	0	1	9
16	1	1	0	0	1	6	0	1	0	0	1	0	0
17	0	1	1	0	1	7	1	0	0	1	0	0	1
18	0	1	2	0	1	8	2	0	0	2	0	0	2
19	0	1	3	0	1	9	3	0	0	3	0	0	3
20	0	1	4	0	1	A	4	0	0	4	0	0	4
21	0	1	5	0	1	B	5	0	0	5	0	0	5
22	0	1	6	0	1	C	6	0	0	6	0	0	6
23	0	1	7	0	1	D	7	0	0	7	0	0	7
24	0	1	8	0	1	E	8	0	0	8	0	0	8
25	0	1	9	0	1	F	9	0	0	9	0	0	9
26	0	1	A	1	1	0	10	0	0	A	0	0	A
27	0	1	B	0	1	1	11	0	0	B	0	0	B
28	0	1	C	0	1	2	12	0	0	C	0	0	C
29	0	1	D	0	1	3	13	0	0	D	0	0	D
30	0	1	E	0	1	4	14	0	0	E	0	0	E
31	0	1	F	0	1	5	15	0	0	F	0	0	F

Hexadecimal operations will not always produce the correct result if performed in decimal mode.

Note that:

- An add instruction with carry (for example, ADC XH,*i*) which uses index registers XH, XL, YH and YL, does not involve decimal correction even if it is performed in the decimal mode. This is because it uses an 8-bit field for 4-bit data.
- The results of the compare instruction (CP) is not decimal-corrected, because the carry flag is ignored.
- The result of the register memory increment instruction (INC M*n*) and decrement instruction (DEC M*n*) are not decimal-corrected.

2.3.2 A and B registers

The A and B registers are 4-bit general-purpose registers used as accumulators. They transfer data and perform ALU operations with other registers, data memory and immediate data.

The data in A can be paired with that in B for use as an indirect jump address by the JPBA instruction.

2.4 Timing Generator

E0C6200/6200A instructions can be divided into three different types depending on the number of clock cycles per instruction: 5, 7 or 12 clock cycles. The more complex the instruction, the more cycles it requires. Note that the number of clock cycles determines the duration of instructions which, in turn, will affect any timing performed in software.

As shown in Figure 2.4.1, the first state of all instructions is a fetch cycle. This is followed by a number of execute cycles.

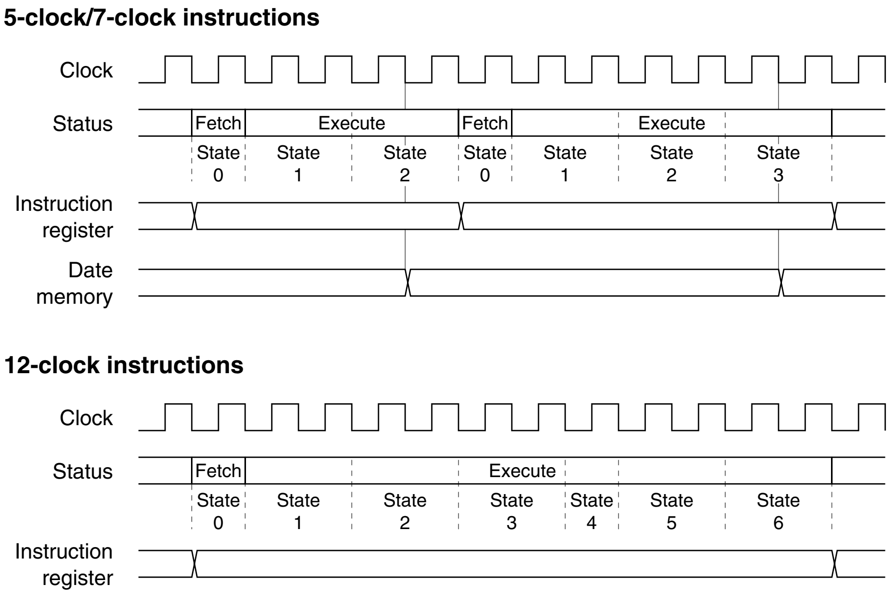


Fig. 2.4.1 Instruction execution timing

2.4.1 HALT and SLP (sleep) modes

HALT and SLP cause the CPU to store the return address on the stack and then stop. HALT will only stop the CPU; the system clock will continue to run. SLP also stops the system clock, resulting in reduced power consumption. The CPU can be restarted by an interrupt.

As interrupts are not automatically enabled by the execution of HALT or SLP, programs should always enable interrupts before executing HALT or SLP, otherwise they will hang waiting for an interrupt.

2.5 Interrupts

The E0C6200/6200A can have up to 15 interrupt vectors. When used with peripheral circuits, these allow internal and external interrupts to be processed easily. See Figure 2.5.3.1 through 2.5.3.4.

2.5.1 Interrupt vectors

The interrupt vectors are assigned to steps 1 to 15 in page 1 of each bank of the program memory. When an interrupt occurs, the program jumps to the appropriate interrupt vector in the current bank.

The priority and linking of these vectors to actual outside events depends on the configuration of the peripheral circuits and therefore is device-specific. This information can be found in the technical manuals for the specific device.

2.5.2 I (interrupt) flag

The I (interrupt) flag enables or disables all interrupts.

When DI or RST F is used to reset the I flag, interrupts are disabled with that instruction step. When EI or SET F is used to set the I flag, interrupts are enabled after the following instruction step. For example, to return control from the interrupt subroutine to the main routine, the sequence EI, RET, does not enable interrupts until after RET has been executed.

The I flag is reset to 0 (DI) on reset.

2.5.3 Operation during interrupt generation

When an interrupt is generated, the program is halted, the program counter (PCP and PCS) is stored on the stack, the I flag is reset to DI mode and NPP is set to 1. The program then branches to the interrupt vector corresponding to the interrupt request. Registers and flags are unaffected by an interrupt.

Register and flag data must be saved by the program since they are not automatically stored on the stack.

The I flag can be set to 1 (EI) within the interrupt subroutine, because nesting of multiple interrupts is available.

If an interrupt is generated while the CPU is in HALT or SLP mode, the CPU is restarted and the interrupt serviced. When the interrupt service routine is completed, the program resumes from the instruction following the HALT or SLP.

<Differences between E0C6200 and E0C6200A>

In the E0C6200 and the E0C6200A, the time it takes to complete interrupt processing by hardware after the Core CPU receives the interrupt request is different as follows:

Table 2.5.3.1 Required interrupt processing time

Item		E0C6200A (clock cycles)	E0C6200 (clock cycles)
a) During instruction execution	12-cycle instruction execution	12.5 to 24.5	13 to 25
	7-cycle instruction execution	12.5 to 19.5	13 to 20
	5-cycle instruction execution	12.5 to 17.5	13 to 18
b) At HALT mode		14 to 15	14 to 15
c) During PSET instruction execution	PSET + CALL	12.5 to 24.5	13 to 25
	PSET + JP	12.5 to 22.5	13 to 23

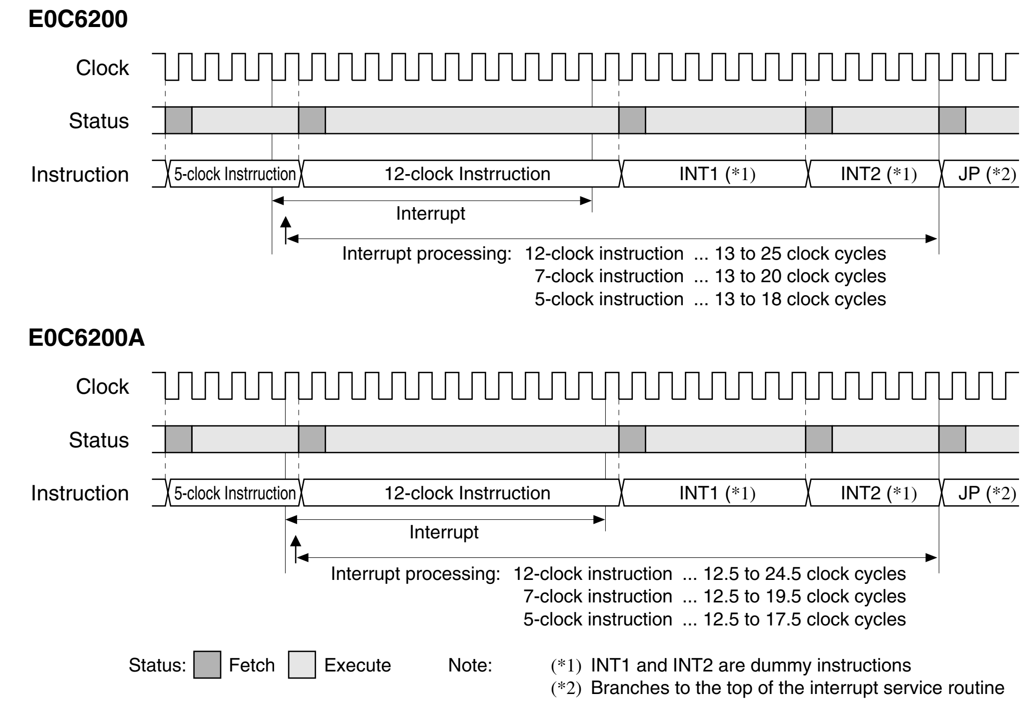


Fig. 2.5.3.1 Interrupt timing during execution

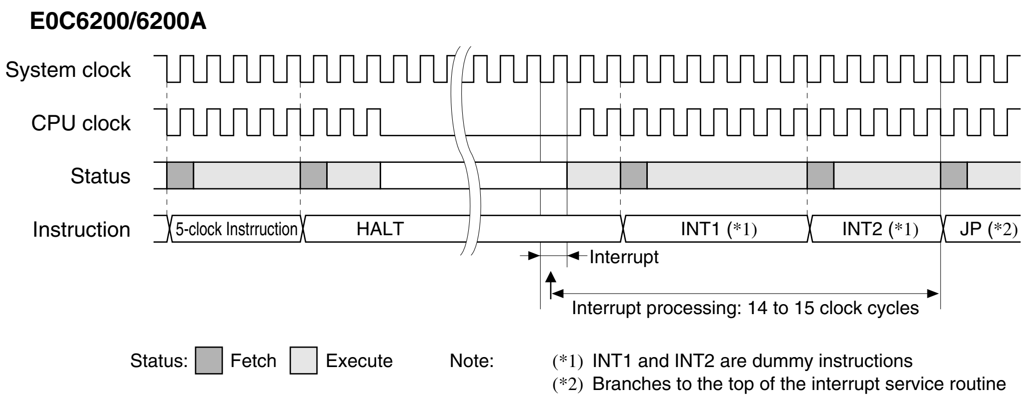
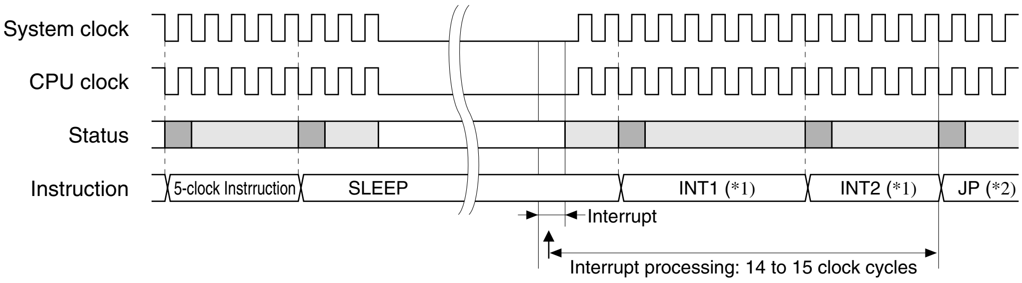


Fig. 2.5.3.2 Interrupt timing in the HALT mode

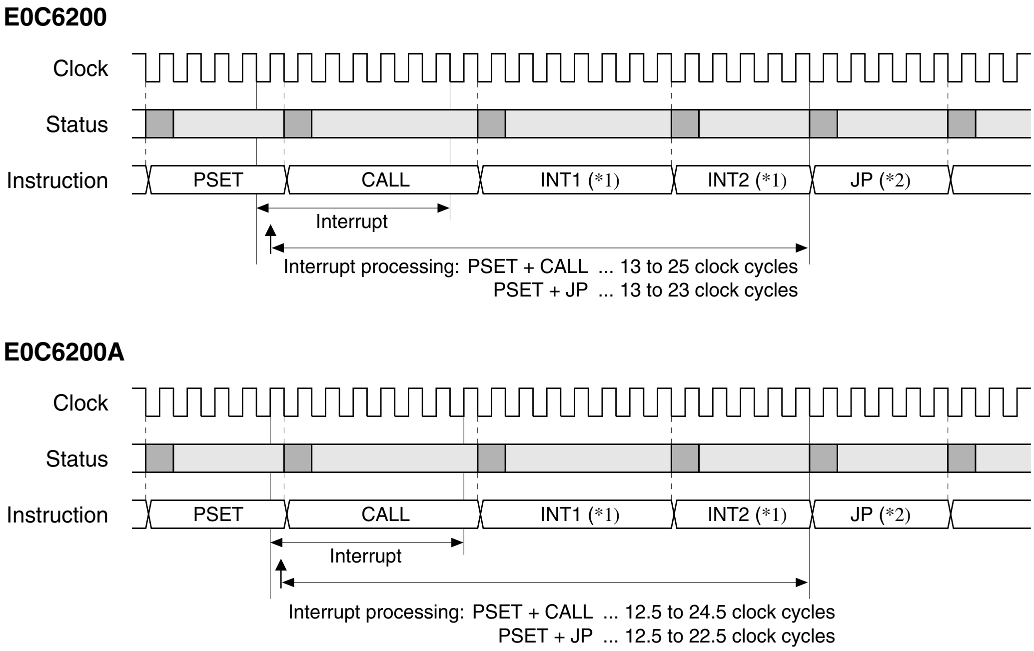
E0C6200/6200A



Status: Fetch Execute

Note: (*1) INT1 and INT2 are dummy instructions
(*2) Branches to the top of the interrupt service routine

Fig. 2.5.3.3 Interrupt timing in SLEEP mode



Status: Fetch Execute

Note: (*1) INT1 and INT2 are dummy instructions
(*2) Branches to the top of the interrupt service routine

Fig. 2.5.3.4 Interrupt timing with PSET

2.5.4 Initial reset

On reset, the registers and flags are set as shown in Table 2.5.4.1.

Table 2.5.4.1 Reset value

		Bit length	Value
Program Counter Step	PCS	8	00H
Program Counter Page	PCP	4	01H
Program Counter Bank	PCB	1	00H
New Page Pointer	NPP	4	01H
New Bank Pointer	NBP	1	Undefined
Stack Pointer	SP	8	Undefined
Index Register	IX	12	Undefined
Index Register	IY	12	Undefined
Register Pointer	RP	4	Undefined
General Register	A	4	Undefined
General Register	B	4	Undefined
Interrupt Flag	I	1	0H
Decimal Flag	D	1	*
Zero Flag	Z	1	Undefined
Carry Flag	C	1	Undefined

* E0C6200 ...Undefined
E0C6200A ...0

<Difference between E0C6200 and E0C6200A>

There is a difference in the setting value of the D (decimal) flag at initial reset between the E0C6200 and the E0C6200A.

Table 2.5.4.2 D (decimal) flag initial setting

CPU Core	E0C6200A	E0C6200
D (decimal) flag setting	0	Undefined

When using the model loaded with the E0C6200 Core CPU, set or reset the D flag in the user's initial routine before using an arithmetic instruction. (refer to the SDF and RDF instructions.)

3 INSTRUCTION SET

This chapter describes the entire instruction set of the E0C6200/6200A Core CPU.
A subset is allocated to each device within the E0C62 Family according to the configuration of the device. Therefore not all instructions are available in every device. The relevant information is in the technical manual for each device.

The source format and a description of the assembler is in the series-specific cross assembler manuals.

The instruction set contains 109 instructions. Each instruction comprises of one 12-bit word.

3.1 Instruction Indices

Three index tables are used for easy reference instructions.

a. Index by function

The instructions are arranged by function.

1. Branch
2. System control
3. Flag operation
4. Stack operation
5. Index operation
6. Data transfer
7. Arithmetic and logical operation

b. Index in alphabetical order

The instructions are arranged in alphabetical order. Page number references are provided.

c. Index by operation code

The instructions are arranged in numerical order by operation code.

3.1.1 By function

Classification	Mne-monic	Operand	Operation Code												Flag				Clock	Operation
			B	A	9	8	7	6	5	4	3	2	1	0	I	D	Z	C		
Branch instructions	PSET	p	1	1	1	0	0	1	0	p4	p3	p2	p1	p0					5	NBP←p4, NPP←p3~p0
	JP	s	0	0	0	0	s7	s6	s5	s4	s3	s2	s1	s0					5	PCB←NBP, PCP←NPP, PCS←s7~s0
		C, s	0	0	1	0	s7	s6	s5	s4	s3	s2	s1	s0					5	PCB←NBP, PCP←NPP, PCS←s7~s0 if C=1
		NC, s	0	0	1	1	s7	s6	s5	s4	s3	s2	s1	s0					5	PCB←NBP, PCP←NPP, PCS←s7~s0 if C=0
		Z, s	0	1	1	0	s7	s6	s5	s4	s3	s2	s1	s0					5	PCB←NBP, PCP←NPP, PCS←s7~s0 if Z=1
		NZ, s	0	1	1	1	s7	s6	s5	s4	s3	s2	s1	s0					5	PCB←NBP, PCP←NPP, PCS←s7~s0 if Z=0
	JPBA		1	1	1	1	1	1	1	0	1	0	0	0					5	PCB←NBP, PCP←NPP, PCSH←B, PCSL←A
	CALL	s	0	1	0	0	s7	s6	s5	s4	s3	s2	s1	s0					7	M(SP-1)←PCP, M(SP-2)←PCSH, M(SP-3)←PCSL+1 SP←SP-3, PCP←NPP, PCS←s7~s0
	CALZ	s	0	1	0	1	s7	s6	s5	s4	s3	s2	s1	s0					7	M(SP-1)←PCP, M(SP-2)←PCSH, M(SP-3)←PCSL+1 SP←SP-3, PCP←0, PCS←s7~s0
	RET		1	1	1	1	1	1	0	1	1	1	1	1					7	PCSL←M(SP), PCSH←M(SP+1), PCP←M(SP+2) SP←SP+3
	RETS		1	1	1	1	1	1	0	1	1	1	1	0					12	PCSL←M(SP), PCSH←M(SP+1), PCP←M(SP+2) SP←SP+3, PC←PC+1
	RETD	e	0	0	0	1	e7	e6	e5	e4	e3	e2	e1	e0					12	PCSL←M(SP), PCSH←M(SP+1), PCP←M(SP+2) SP←SP+3, M(X)←e3~e0, M(X+1)←e7~e4, X←X+2
System control instructions	NOP5		1	1	1	1	1	1	1	1	1	0	1	1					5	No operation (5 clock cycles)
	NOP7		1	1	1	1	1	1	1	1	1	1	1	1					7	No operation (7 clock cycles)
	HALT		1	1	1	1	1	1	1	1	1	0	0	0					5	Halt (stop clock)
	SLP		1	1	1	1	1	1	1	1	1	0	0	1					5	SLEEP (stop oscillation)
Index operation instructions	INC	X	1	1	1	0	1	1	1	0	0	0	0	0					5	X←X+1
		Y	1	1	1	0	1	1	1	1	0	0	0	0					5	Y←Y+1
	LD	X, e	1	0	1	1	e7	e6	e5	e4	e3	e2	e1	e0					5	XH←e7~e4, XL←e3~e0
		Y, e	1	0	0	0	e7	e6	e5	e4	e3	e2	e1	e0					5	YH←e7~e4, YL←e3~e0
		XP, r	1	1	1	0	1	0	0	0	0	0	r1	r0					5	XP←r
		XH, r	1	1	1	0	1	0	0	0	0	1	r1	r0					5	XH←r
		XL, r	1	1	1	0	1	0	0	0	1	0	r1	r0					5	XL←r
		YP, r	1	1	1	0	1	0	0	1	0	0	r1	r0					5	YP←r
		YH, r	1	1	1	0	1	0	0	1	0	1	r1	r0					5	YH←r
		YL, r	1	1	1	0	1	0	0	1	1	0	r1	r0					5	YL←r
		r, XP	1	1	1	0	1	0	1	0	0	0	r1	r0					5	r←XP
		r, XH	1	1	1	0	1	0	1	0	0	1	r1	r0					5	r←XH
		r, XL	1	1	1	0	1	0	1	0	1	0	r1	r0					5	r←XL
		r, YP	1	1	1	0	1	0	1	1	0	0	r1	r0					5	r←YP
		r, YH	1	1	1	0	1	0	1	1	0	1	r1	r0					5	r←YH
		r, YL	1	1	1	0	1	0	1	1	1	0	r1	r0					5	r←YL
	ADC	XH, i	1	0	1	0	0	0	0	0	i3	i2	i1	i0			↕	↕	7	XH←XH+i3~i0+C
		XL, i	1	0	1	0	0	0	0	1	i3	i2	i1	i0			↕	↕	7	XL←XL+i3~i0+C
		YH, i	1	0	1	0	0	0	1	0	i3	i2	i1	i0			↕	↕	7	YH←YH+i3~i0+C
		YL, i	1	0	1	0	0	0	1	1	i3	i2	i1	i0			↕	↕	7	YL←YL+i3~i0+C

Classification	Mne-monic	Operand	Operation Code												Flag				Clock	Operation
			B	A	9	8	7	6	5	4	3	2	1	0	I	D	Z	C		
Index operation instructions	CP	XH, i	1	0	1	0	0	1	0	0	i3	i2	i1	i0			↕	↕	7	XH-i3~i0
		XL, i	1	0	1	0	0	1	0	1	i3	i2	i1	i0			↕	↕	7	XL-i3~i0
		YH, i	1	0	1	0	0	1	1	0	i3	i2	i1	i0			↕	↕	7	YH-i3~i0
		YL, i	1	0	1	0	0	1	1	1	i3	i2	i1	i0			↕	↕	7	YL-i3~i0
Data transfer instructions	LD	r, i	1	1	1	0	0	0	r1	r0	i3	i2	i1	i0					5	r←i3~i0
		r, q	1	1	1	0	1	1	0	0	r1	r0	q1	q0					5	r←q
		A, Mn	1	1	1	1	1	0	1	0	n3	n2	n1	n0					5	A←M(n3~n0)
		B, Mn	1	1	1	1	1	0	1	1	n3	n2	n1	n0					5	B←M(n3~n0)
		Mn, A	1	1	1	1	1	0	0	0	n3	n2	n1	n0					5	M(n3~n0)←A
		Mn, B	1	1	1	1	1	0	0	1	n3	n2	n1	n0					5	M(n3~n0)←B
	LDPX	MX, i	1	1	1	0	0	1	1	0	i3	i2	i1	i0					5	M(X)←i3~i0, X←X+1
		r, q	1	1	1	0	1	1	1	0	r1	r0	q1	q0					5	r←q, X←X+1
	LDPY	MY, i	1	1	1	0	0	1	1	1	i3	i2	i1	i0					5	M(Y)←i3~i0, Y←Y+1
		r, q	1	1	1	0	1	1	1	1	r1	r0	q1	q0					5	r←q, Y←Y+1
	LBPX	MX, e	1	0	0	1	e7	e6	e5	e4	e3	e2	e1	e0					5	M(X)←e3~e0, M(X+1)←e7~e4, X←X+2
Flag operation instructions	SET	F, i	1	1	1	1	0	1	0	0	i3	i2	i1	i0	↑	↑	↑	↑	7	F←F∨i3~i0
	RST	F, i	1	1	1	1	0	1	0	1	i3	i2	i1	i0	↓	↓	↓	↓	7	F←F∧i3~i0
	SCF		1	1	1	1	0	1	0	0	0	0	0	1				↑	7	C←1
	RCF		1	1	1	1	0	1	0	1	1	1	1	0				↓	7	C←0
	SZF		1	1	1	1	0	1	0	0	0	0	1	0			↑		7	Z←1
	RZF		1	1	1	1	0	1	0	1	1	1	0	1			↓		7	Z←0
	SDF		1	1	1	1	0	1	0	0	0	1	0	0		↑			7	D←1 (Decimal Adjuster ON)
	RDF		1	1	1	1	0	1	0	1	1	0	1	1		↓			7	D←0 (Decimal Adjuster OFF)
	EI		1	1	1	1	0	1	0	0	1	0	0	0	↑				7	I←1 (Enables Interrupt)
	DI		1	1	1	1	0	1	0	1	0	1	1	1	↓				7	I←0 (Disables Interrupt)
Stack operation instructions	INC	SP	1	1	1	1	1	1	0	1	1	0	1	1					5	SP←SP+1
	DEC	SP	1	1	1	1	1	1	0	0	1	0	1	1					5	SP←SP-1
	PUSH	r	1	1	1	1	1	1	0	0	0	0	r1	r0					5	SP←SP-1, M(SP)←r
		XP	1	1	1	1	1	1	0	0	0	1	0	0					5	SP←SP-1, M(SP)←XP
		XH	1	1	1	1	1	1	0	0	0	1	0	1					5	SP←SP-1, M(SP)←XH
		XL	1	1	1	1	1	1	0	0	0	1	1	0					5	SP←SP-1, M(SP)←XL
		YP	1	1	1	1	1	1	0	0	0	1	1	1					5	SP←SP-1, M(SP)←YP
		YH	1	1	1	1	1	1	0	0	1	0	0	0					5	SP←SP-1, M(SP)←YH
		YL	1	1	1	1	1	1	0	0	1	0	0	1					5	SP←SP-1, M(SP)←YL
		F	1	1	1	1	1	1	0	0	1	0	1	0					5	SP←SP-1, M(SP)←F
	POP	r	1	1	1	1	1	1	0	1	0	0	r1	r0					5	r←M(SP), SP←SP+1
		XP	1	1	1	1	1	1	0	1	0	1	0	0					5	XP←M(SP), SP←SP+1
		XH	1	1	1	1	1	1	0	1	0	1	0	1					5	XH←M(SP), SP←SP+1
		XL	1	1	1	1	1	1	0	1	0	1	1	0					5	XL←M(SP), SP←SP+1
		YP	1	1	1	1	1	1	0	1	0	1	1	1					5	YP←M(SP), SP←SP+1

Classification	Mne-monic	Operand	Operation Code												Flag				Clock	Operation
			B	A	9	8	7	6	5	4	3	2	1	0	I	D	Z	C		
Stack operation instructions	POP	YH	1	1	1	1	1	1	0	1	1	0	0	0					5	YH←M(SP), SP←SP+1
		YL	1	1	1	1	1	1	0	1	1	0	0	1					5	YL←M(SP), SP←SP+1
		F	1	1	1	1	1	1	0	1	1	0	1	0	↕	↕	↕	↕	5	F←M(SP), SP←SP+1
	LD	SPH, r	1	1	1	1	1	1	1	0	0	0	r1	r0					5	SPH←r
		SPL, r	1	1	1	1	1	1	1	1	0	0	r1	r0					5	SPL←r
		r, SPH	1	1	1	1	1	1	1	0	0	1	r1	r0					5	r←SPH
		r, SPL	1	1	1	1	1	1	1	1	0	1	r1	r0					5	r←SPL
Arithmetic instructions	ADD	r, i	1	1	0	0	0	0	r1	r0	i3	i2	i1	i0		★	↕	↕	7	r←r+i3~i0
		r, q	1	0	1	0	1	0	0	0	r1	r0	q1	q0		★	↕	↕	7	r←r+q
	ADC	r, i	1	1	0	0	0	1	r1	r0	i3	i2	i1	i0		★	↕	↕	7	r←r+i3~i0+C
		r, q	1	0	1	0	1	0	0	1	r1	r0	q1	q0		★	↕	↕	7	r←r+q+C
	SUB	r, q	1	0	1	0	1	0	1	0	r1	r0	q1	q0		★	↕	↕	7	r←r-q
	SBC	r, i	1	1	0	1	0	1	r1	r0	i3	i2	i1	i0		★	↕	↕	7	r←r-i3~i0-C
		r, q	1	0	1	0	1	0	1	1	r1	r0	q1	q0		★	↕	↕	7	r←r-q-C
	AND	r, i	1	1	0	0	1	0	r1	r0	i3	i2	i1	i0			↕		7	r←r∧i3~i0
		r, q	1	0	1	0	1	1	0	0	r1	r0	q1	q0			↕		7	r←r∧q
	OR	r, i	1	1	0	0	1	1	r1	r0	i3	i2	i1	i0			↕		7	r←r∨i3~i0
		r, q	1	0	1	0	1	1	0	1	r1	r0	q1	q0			↕		7	r←r∨q
	XOR	r, i	1	1	0	1	0	0	r1	r0	i3	i2	i1	i0			↕		7	r←r∀i3~i0
		r, q	1	0	1	0	1	1	1	0	r1	r0	q1	q0			↕		7	r←r∀q
	CP	r, i	1	1	0	1	1	1	r1	r0	i3	i2	i1	i0			↕	↕	7	r-i3~i0
		r, q	1	1	1	1	0	0	0	0	r1	r0	q1	q0			↕	↕	7	r-q
	FAN	r, i	1	1	0	1	1	0	r1	r0	i3	i2	i1	i0			↕		7	r∧i3~i0
		r, q	1	1	1	1	0	0	0	1	r1	r0	q1	q0			↕		7	r∧q
	RLC	r	1	0	1	0	1	1	1	1	r1	r0	r1	r0			↕	↕	7	d3←d2, d2←d1, d1←d0, d0←C, C←d3
	RRC	r	1	1	1	0	1	0	0	0	1	1	r1	r0			↕	↕	5	d3←C, d2←d3, d1←d2, d0←d1, C←d0
	INC	Mn	1	1	1	1	0	1	1	0	n3	n2	n1	n0			↕	↕	7	M(n3~n0)←M(n3~n0)+1
	DEC	Mn	1	1	1	1	0	1	1	1	n3	n2	n1	n0			↕	↕	7	M(n3~n0)←M(n3~n0)-1
	ACPX	MX, r	1	1	1	1	0	0	1	0	1	0	r1	r0		★	↕	↕	7	M(X)←M(X)+r+C, X←X+1
	ACPY	MY, r	1	1	1	1	0	0	1	0	1	1	r1	r0		★	↕	↕	7	M(Y)←M(Y)+r+C, Y←Y+1
	SCPX	MX, r	1	1	1	1	0	0	1	1	1	0	r1	r0		★	↕	↕	7	M(X)←M(X)-r-C, X←X+1
	SCPY	MY, r	1	1	1	1	0	0	1	1	1	1	r1	r0		★	↕	↕	7	M(Y)←M(Y)-r-C, Y←Y+1
	NOT	r	1	1	0	1	0	0	r1	r0	1	1	1	1			↕		7	$r \leftarrow \overline{r}$

3.1.2 In alphabetical order

Page	Mne-monic	Operand	Operation Code												Flag				Clock	Operation
			B	A	9	8	7	6	5	4	3	2	1	0	I	D	Z	C		
28	ACPX	MX, r	1	1	1	1	0	0	1	0	1	0	r1	r0		★	↕	↕	7	M(X)←M(X)+r+C, X←X+1
28	ACPY	MY, r	1	1	1	1	0	0	1	0	1	1	r1	r0		★	↕	↕	7	M(Y)←M(Y)+r+C, Y←Y+1
29	ADC	r, i	1	1	0	0	0	1	r1	r0	i3	i2	i1	i0		★	↕	↕	7	r←r+i3~i0+C
29		r, q	1	0	1	0	1	0	0	1	r1	r0	q1	q0		★	↕	↕	7	r←r+q+C
30		XH, i	1	0	1	0	0	0	0	0	i3	i2	i1	i0			↕	↕	7	XH←XH+i3~i0+C
30		XL, i	1	0	1	0	0	0	0	1	i3	i2	i1	i0			↕	↕	7	XL←XL+i3~i0+C
31		YH, i	1	0	1	0	0	0	1	0	i3	i2	i1	i0			↕	↕	7	YH←YH+i3~i0+C
31		YL, i	1	0	1	0	0	0	1	1	i3	i2	i1	i0			↕	↕	7	YL←YL+i3~i0+C
32	ADD	r, i	1	1	0	0	0	0	r1	r0	i3	i2	i1	i0		★	↕	↕	7	r←r+i3~i0
32		r, q	1	0	1	0	1	0	0	0	r1	r0	q1	q0		★	↕	↕	7	r←r+q
33	AND	r, i	1	1	0	0	1	0	r1	r0	i3	i2	i1	i0			↕		7	r←r∧i3~i0
33		r, q	1	0	1	0	1	1	0	0	r1	r0	q1	q0			↕		7	r←r∧q
34	CALL	s	0	1	0	0	s7	s6	s5	s4	s3	s2	s1	s0					7	M(SP-1)←PCP, M(SP-2)←PCSH, M(SP-3)←PCSL+1 SP←SP-3, PCP←NPP, PCS←s7~s0
34	CALZ	s	0	1	0	1	s7	s6	s5	s4	s3	s2	s1	s0					7	M(SP-1)←PCP, M(SP-2)←PCSH, M(SP-3)←PCSL+1 SP←SP-3, PCP←0, PCS←s7~s0
35	CP	r, i	1	1	0	1	1	1	r1	r0	i3	i2	i1	i0			↕	↕	7	r-i3~i0
35		r, q	1	1	1	1	0	0	0	0	r1	r0	q1	q0			↕	↕	7	r-q
36		XH, i	1	0	1	0	0	1	0	0	i3	i2	i1	i0			↕	↕	7	XH-i3~i0
36		XL, i	1	0	1	0	0	1	0	1	i3	i2	i1	i0			↕	↕	7	XL-i3~i0
37		YH, i	1	0	1	0	0	1	1	0	i3	i2	i1	i0			↕	↕	7	YH-i3~i0
37		YL, i	1	0	1	0	0	1	1	1	i3	i2	i1	i0			↕	↕	7	YL-i3~i0
38	DEC	Mn	1	1	1	1	0	1	1	1	n3	n2	n1	n0			↕	↕	7	M(n3~n0)←M(n3~n0)-1
38		SP	1	1	1	1	1	1	0	0	1	0	1	1					5	SP←SP-1
39	DI		1	1	1	1	0	1	0	1	0	1	1	1	↓				7	I←0 (Disables Interrupt)
39	EI		1	1	1	1	0	1	0	0	1	0	0	0	↑				7	I←1 (Enables Interrupt)
40	FAN	r, i	1	1	0	1	1	0	r1	r0	i3	i2	i1	i0			↕		7	r∧i3~i0
40		r, q	1	1	1	1	0	0	0	1	r1	r0	q1	q0			↕		7	r∧q
41	HALT		1	1	1	1	1	1	1	1	1	0	0	0					5	Halt (stop clock)
41	INC	Mn	1	1	1	1	0	1	1	0	n3	n2	n1	n0			↕	↕	7	M(n3~n0)←M(n3~n0)+1
42		SP	1	1	1	1	1	1	0	1	1	0	1	1					5	SP←SP+1
42		X	1	1	1	0	1	1	1	0	0	0	0	0					5	X←X+1
43		Y	1	1	1	0	1	1	1	1	0	0	0	0					5	Y←Y+1
43	JPBA		1	1	1	1	1	1	1	0	1	0	0	0					5	PCB←NBP, PCP←NPP, PCSH←B, PCSL←A
44	JP	C, s	0	0	1	0	s7	s6	s5	s4	s3	s2	s1	s0					5	PCB←NBP, PCP←NPP, PCS←s7~s0 if C=1
44		NC, s	0	0	1	1	s7	s6	s5	s4	s3	s2	s1	s0					5	PCB←NBP, PCP←NPP, PCS←s7~s0 if C=0
45		NZ, s	0	1	1	1	s7	s6	s5	s4	s3	s2	s1	s0					5	PCB←NBP, PCP←NPP, PCS←s7~s0 if Z=0
45		s	0	0	0	0	s7	s6	s5	s4	s3	s2	s1	s0					5	PCB←NBP, PCP←NPP, PCS←s7~s0
46		Z, s	0	1	1	0	s7	s6	s5	s4	s3	s2	s1	s0					5	PCB←NBP, PCP←NPP, PCS←s7~s0 if Z=1
46	LBPX	MX, e	1	0	0	1	e7	e6	e5	e4	e3	e2	e1	e0					5	M(X)←e3~e0, M(X+1)←e7~e4, X←X+2

Page	Mne-monic	Operand	Operation Code												Flag				Clock	Operation
			B	A	9	8	7	6	5	4	3	2	1	0	I	D	Z	C		
47	LD	A, Mn	1	1	1	1	1	0	1	0	n3	n2	n1	n0					5	A←M(n3~n0)
47		B, Mn	1	1	1	1	1	0	1	1	n3	n2	n1	n0					5	B←M(n3~n0)
48		Mn, A	1	1	1	1	1	0	0	0	n3	n2	n1	n0					5	M(n3~n0)←A
48		Mn, B	1	1	1	1	1	0	0	1	n3	n2	n1	n0					5	M(n3~n0)←B
51		r, i	1	1	1	0	0	0	r1	r0	i3	i2	i1	i0					5	r←i3~i0
51		r, q	1	1	1	0	1	1	0	0	r1	r0	q1	q0					5	r←q
52		r, SPH	1	1	1	1	1	1	1	0	0	1	r1	r0					5	r←SPH
52		r, SPL	1	1	1	1	1	1	1	1	0	1	r1	r0					5	r←SPL
53		r, XH	1	1	1	0	1	0	1	0	0	1	r1	r0					5	r←XH
53		r, XL	1	1	1	0	1	0	1	0	1	0	r1	r0					5	r←XL
54		r, XP	1	1	1	0	1	0	1	0	0	0	r1	r0					5	r←XP
54		r, YH	1	1	1	0	1	0	1	1	0	1	r1	r0					5	r←YH
55		r, YL	1	1	1	0	1	0	1	1	1	0	r1	r0					5	r←YL
55		r, YP	1	1	1	0	1	0	1	1	0	0	r1	r0					5	r←YP
56		SPH, r	1	1	1	1	1	1	1	0	0	0	r1	r0					5	SPH←r
56		SPL, r	1	1	1	1	1	1	1	1	0	0	r1	r0					5	SPL←r
57		XH, r	1	1	1	0	1	0	0	0	0	1	r1	r0					5	XH←r
58		XL, r	1	1	1	0	1	0	0	0	1	0	r1	r0					5	XL←r
58		XP, r	1	1	1	0	1	0	0	0	0	0	r1	r0					5	XP←r
57		X, e	1	0	1	1	e7	e6	e5	e4	e3	e2	e1	e0					5	XH←e7~e4, XL←e3~e0
59		YH, r	1	1	1	0	1	0	0	1	0	1	r1	r0					5	YH←r
60		YL, r	1	1	1	0	1	0	0	1	1	0	r1	r0					5	YL←r
60		YP, r	1	1	1	0	1	0	0	1	0	0	r1	r0					5	YP←r
59		Y, e	1	0	0	0	e7	e6	e5	e4	e3	e2	e1	e0					5	YH←e7~e4, YL←e3~e0
49	LDPX	MX, i	1	1	1	0	0	1	1	0	i3	i2	i1	i0					5	M(X)←i3~i0, X←X+1
49		r, q	1	1	1	0	1	1	1	0	r1	r0	q1	q0					5	r←q, X ← X+1
50	LDPY	MY, i	1	1	1	0	0	1	1	1	i3	i2	i1	i0					5	M(Y)←i3~i0, Y←Y+1
50		r, q	1	1	1	0	1	1	1	1	r1	r0	q1	q0					5	r←q, Y←Y+1
61	NOP5		1	1	1	1	1	1	1	1	1	0	1	1					5	No operation (5 clock cycles)
61	NOP7		1	1	1	1	1	1	1	1	1	1	1	1					7	No operation (7 clock cycles)
62	NOT	r	1	1	0	1	0	0	r1	r0	1	1	1	1			↕		7	r←$\overline{r}$
62	OR	r, i	1	1	0	0	1	1	r1	r0	i3	i2	i1	i0			↕		7	r←r∨i3~i0
63		r, q	1	0	1	0	1	1	0	1	r1	r0	q1	q0			↕		7	r←r∨q
63	POP	F	1	1	1	1	1	1	0	1	1	0	1	0	↕	↕	↕	↕	5	F←M(SP), SP←SP+1
64		r	1	1	1	1	1	1	0	1	0	0	r1	r0					5	r←M(SP), SP←SP+1
64		XH	1	1	1	1	1	1	0	1	0	1	0	1					5	XH←M(SP), SP←SP+1
65		XL	1	1	1	1	1	1	0	1	0	1	1	0					5	XL←M(SP), SP←SP+1
65		XP	1	1	1	1	1	1	0	1	0	1	0	0					5	XP←M(SP), SP←SP+1
66		YH	1	1	1	1	1	1	0	1	1	0	0	0					5	YH←M(SP), SP←SP+1
66		YL	1	1	1	1	1	1	0	1	1	0	0	1					5	YL←M(SP), SP←SP+1
67		YP	1	1	1	1	1	1	0	1	0	1	1	1					5	YP←M(SP), SP←SP+1

Page	Mne-monic	Operand	Operation Code												Flag				Clock	Operation
			B	A	9	8	7	6	5	4	3	2	1	0	I	D	Z	C		
67	PSET	p	1	1	1	0	0	1	0	p4	p3	p2	p1	p0					5	NBP←p4, NPP←p3~p0
68	PUSH	F	1	1	1	1	1	1	0	0	1	0	1	0					5	SP←SP-1, M(SP)←F
68		r	1	1	1	1	1	1	0	0	0	0	r1	r0					5	SP←SP-1, M(SP)←r
69		XH	1	1	1	1	1	1	0	0	0	1	0	1					5	SP←SP-1, M(SP)←XH
69		XL	1	1	1	1	1	1	0	0	0	1	1	0					5	SP←SP-1, M(SP)←XL
70		XP	1	1	1	1	1	1	0	0	0	1	0	0					5	SP←SP-1, M(SP)←XP
70		YH	1	1	1	1	1	1	0	0	1	0	0	0					5	SP←SP-1, M(SP)←YH
71		YL	1	1	1	1	1	1	0	0	1	0	0	1					5	SP←SP-1, M(SP)←YL
71		YP	1	1	1	1	1	1	0	0	0	1	1	1					5	SP←SP-1, M(SP)←YP
72	RCF		1	1	1	1	0	1	0	1	1	1	1	0				↓	7	C←0
72	RDF		1	1	1	1	0	1	0	1	1	0	1	1		↓			7	D←0 (Decimal Adjuster OFF)
73	RET		1	1	1	1	1	1	0	1	1	1	1	1					7	PCSL←M(SP), PCSH←M(SP+1), PCP←M(SP+2) SP←SP+3
73	RETD	e	0	0	0	1	e7	e6	e5	e4	e3	e2	e1	e0					12	PCSL←M(SP), PCSH←M(SP+1), PCP←M(SP+2) SP←SP+3, M(X)←e3~e0, M(X+1)←e7~e4, X←X+2
74	RETS		1	1	1	1	1	1	0	1	1	1	1	0					12	PCSL←M(SP), PCSH←M(SP+1), PCP←M(SP+2) SP←SP+3, PC←PC+1
74	RLC	r	1	0	1	0	1	1	1	1	r1	r0	r1	r0			↕	↕	7	d3←d2, d2←d1, d1←d0, d0←C, C←d3
75	RRC	r	1	1	1	0	1	0	0	0	1	1	r1	r0			↕	↕	5	d3←C, d2←d3, d1←d2, d0←d1, C←d0
75	RST	F, i	1	1	1	1	0	1	0	1	i3	i2	i1	i0	↓	↓	↓	↓	7	F←F∧i3~i0
76	RZF		1	1	1	1	0	1	0	1	1	1	0	1			↓		7	Z←0
76	SBC	r, i	1	1	0	1	0	1	r1	r0	i3	i2	i1	i0		★	↕	↕	7	r←r-i3~i0-C
77		r, q	1	0	1	0	1	0	1	1	r1	r0	q1	q0		★	↕	↕	7	r←r-q-C
77	SCF		1	1	1	1	0	1	0	0	0	0	0	1				↑	7	C←1
78	SCPX	MX, r	1	1	1	1	0	0	1	1	1	0	r1	r0		★	↕	↕	7	M(X)←M(X)-r-C, X←X+1
78	SCPY	MY, r	1	1	1	1	0	0	1	1	1	1	r1	r0		★	↕	↕	7	M(Y)←M(Y)-r-C, Y←Y+1
79	SDF		1	1	1	1	0	1	0	0	0	1	0	0		↑			7	D←1 (Decimal Adjuster ON)
79	SET	F, i	1	1	1	1	0	1	0	0	i3	i2	i1	i0	↑	↑	↑	↑	7	F←F∨i3~i0
80	SLP		1	1	1	1	1	1	1	1	1	0	0	1					5	SLEEP (stop oscillation)
80	SUB	r, q	1	0	1	0	1	0	1	0	r1	r0	q1	q0		★	↕	↕	7	r←r-q
81	SZF		1	1	1	1	0	1	0	0	0	0	1	0			↑		7	Z←1
81	XOR	r, i	1	1	0	1	0	0	r1	r0	i3	i2	i1	i0			↕		7	r←r∀i3~i0
82		r, q	1	0	1	0	1	1	1	0	r1	r0	q1	q0			↕		7	r←r∀q

3.1.3 By operation code

Operation Code (HEX)	Mne-monic	Operand	Operation Code												Flag				Clock	Operation
			B	A	9	8	7	6	5	4	3	2	1	0	I	D	Z	C		
000 to 0FF	JP	s	0	0	0	0	s7	s6	s5	s4	s3	s2	s1	s0					5	PCB←NBP, PCP←NPP, PCS←s7~s0
100 to 1FF	RETD	e	0	0	0	1	e7	e6	e5	e4	e3	e2	e1	e0					12	PCSL←M(SP), PCSH←M(SP+1), PCP←M(SP+2) SP←SP+3, M(X)←e3~e0, M(X+1)←e7~e4, X←X+2
200 to 2FF	JP	C, s	0	0	1	0	s7	s6	s5	s4	s3	s2	s1	s0					5	PCB←NBP, PCP←NPP, PCS←s7~s0 if C=1
300 to 3FF	JP	NC, s	0	0	1	1	s7	s6	s5	s4	s3	s2	s1	s0					5	PCB←NBP, PCP←NPP, PCS←s7~s0 if C=0
400 to 4FF	CALL	s	0	1	0	0	s7	s6	s5	s4	s3	s2	s1	s0					7	M(SP-1)←PCP, M(SP-2)←PCSH, M(SP-3)←PCSL+1 SP←SP-3, PCP←NPP, PCS←s7~s0
500 to 5FF	CALZ	s	0	1	0	1	s7	s6	s5	s4	s3	s2	s1	s0					7	M(SP-1)←PCP, M(SP-2)←PCSH, M(SP-3)←PCSL+1 SP←SP-3, PCP←0, PCS←s7~s0
600 to 6FF	JP	Z, s	0	1	1	0	s7	s6	s5	s4	s3	s2	s1	s0					5	PCB←NBP, PCP←NPP, PCS←s7~s0 if Z=1
700 to 7FF	JP	NZ, s	0	1	1	1	s7	s6	s5	s4	s3	s2	s1	s0					5	PCB←NBP, PCP←NPP, PCS←s7~s0 if Z=0
800 to 8FF	LD	Y, e	1	0	0	0	e7	e6	e5	e4	e3	e2	e1	e0					5	YH←e7~e4, YL←e3~e0
900 to 9FF	LBPX	MX, e	1	0	0	1	e7	e6	e5	e4	e3	e2	e1	e0					5	M(X)←e3~e0, M(X+1)←e7~e4, X←X+2
A00 to A0F	ADC	XH, i	1	0	1	0	0	0	0	0	i3	i2	i1	i0			↕	↕	7	XH←XH+i3~i0+C
A10 to A1F	ADC	XL, i	1	0	1	0	0	0	0	1	i3	i2	i1	i0			↕	↕	7	XL←XL+i3~i0+C
A20 to A2F	ADC	YH, i	1	0	1	0	0	0	1	0	i3	i2	i1	i0			↕	↕	7	YH←YH+i3~i0+C
A30 to A3F	ADC	YL, i	1	0	1	0	0	0	1	1	i3	i2	i1	i0			↕	↕	7	YL←YL+i3~i0+C
A40 to A4F	CP	XH, i	1	0	1	0	0	1	0	0	i3	i2	i1	i0			↕	↕	7	XH-i3~i0
A50 to A5F	CP	XL, i	1	0	1	0	0	1	0	1	i3	i2	i1	i0			↕	↕	7	XL-i3~i0
A60 to A6F	CP	YH, i	1	0	1	0	0	1	1	0	i3	i2	i1	i0			↕	↕	7	YH-i3~i0
A70 to A7F	CP	YL, i	1	0	1	0	0	1	1	1	i3	i2	i1	i0			↕	↕	7	YL-i3~i0
A80 to A8F	ADD	r, q	1	0	1	0	1	0	0	0	r1	r0	q1	q0		★	↕	↕	7	r←r+q
A90 to A9F	ADC	r, q	1	0	1	0	1	0	0	1	r1	r0	q1	q0		★	↕	↕	7	r←r+q+C
AA0 to AAF	SUB	r, q	1	0	1	0	1	0	1	0	r1	r0	q1	q0		★	↕	↕	7	r←r-q
AB0 to ABF	SBC	r, q	1	0	1	0	1	0	1	1	r1	r0	q1	q0		★	↕	↕	7	r←r-q-C
AC0 to ACF	AND	r, q	1	0	1	0	1	1	0	0	r1	r0	q1	q0			↕		7	r←r∧q
AD0 to ADF	OR	r, q	1	0	1	0	1	1	0	1	r1	r0	q1	q0			↕		7	r←r∨q
AE0 to AEF	XOR	r, q	1	0	1	0	1	1	1	0	r1	r0	q1	q0			↕		7	r←r∀q
AF0 to AFF	RLC	r	1	0	1	0	1	1	1	1	r1	r0	r1	r0			↕	↕	7	d3←d2, d2←d1, d1←d0, d0←C, C←d3
B00 to BFF	LD	X, e	1	0	1	1	e7	e6	e5	e4	e3	e2	e1	e0					5	XH←e7~e4, XL←e3~e0
C00 to C3F	ADD	r, i	1	1	0	0	0	0	r1	r0	i3	i2	i1	i0		★	↕	↕	7	r←r+i3~i0
C40 to C7F	ADC	r, i	1	1	0	0	0	1	r1	r0	i3	i2	i1	i0		★	↕	↕	7	r←r+i3~i0+C
C80 to CBF	AND	r, i	1	1	0	0	1	0	r1	r0	i3	i2	i1	i0			↕		7	r←r∧i3~i0
CC0 to CFF	OR	r, i	1	1	0	0	1	1	r1	r0	i3	i2	i1	i0			↕		7	r←r∨i3~i0
D00 to D3F	XOR	r, i	1	1	0	1	0	0	r1	r0	i3	i2	i1	i0			↕		7	r←r∀i3~i0
D0F to D3F	NOT	r	1	1	0	1	0	0	r1	r0	1	1	1	1			↕		7	r←$\overline{r}$
D40 to D7F	SBC	r, i	1	1	0	1	0	1	r1	r0	i3	i2	i1	i0		★	↕	↕	7	r←r-i3~i0-C
D80 to DBF	FAN	r, i	1	1	0	1	1	0	r1	r0	i3	i2	i1	i0			↕		7	r∧i3~i0
DC0 to DFF	CP	r, i	1	1	0	1	1	1	r1	r0	i3	i2	i1	i0			↕	↕	7	r-i3~i0
E00 to E3F	LD	r, i	1	1	1	0	0	0	r1	r0	i3	i2	i1	i0					5	r←i3~i0

Operation Code (HEX)	Mne-monic	Operand	Operation Code												Flag				Clock	Operation
			B	A	9	8	7	6	5	4	3	2	1	0	I	D	Z	C		
E40 to E5F	PSET	p	1	1	1	0	0	1	0	p4	p3	p2	p1	p0					5	NBP←p4, NPP←p3~p0
E60 to E6F	LDPX	MX, i	1	1	1	0	0	1	1	0	i3	i2	i1	i0					5	M(X)←i3~i0, X←X+1
E70 to E7F	LDPY	MY, i	1	1	1	0	0	1	1	1	i3	i2	i1	i0					5	M(Y)←i3~i0, Y←Y+1
E80 to E83	LD	XP, r	1	1	1	0	1	0	0	0	0	0	r1	r0					5	XP←r
E84 to E87	LD	XH, r	1	1	1	0	1	0	0	0	0	1	r1	r0					5	XH←r
E88 to E8B	LD	XL, r	1	1	1	0	1	0	0	0	1	0	r1	r0					5	XL←r
E8C to E8F	RRC	r	1	1	1	0	1	0	0	0	1	1	r1	r0			↕	↕	5	d3←C, d2←d3, d1←d2, d0←d1, C←d0
E90 to E93	LD	YP, r	1	1	1	0	1	0	0	1	0	0	r1	r0					5	YP←r
E94 to E97	LD	YH, r	1	1	1	0	1	0	0	1	0	1	r1	r0					5	YH←r
E98 to E9B	LD	YL, r	1	1	1	0	1	0	0	1	1	0	r1	r0					5	YL←r
EA0 to EA3	LD	r, XP	1	1	1	0	1	0	1	0	0	0	r1	r0					5	r←XP
EA4 to EA7	LD	r, XH	1	1	1	0	1	0	1	0	0	1	r1	r0					5	r←XH
EA8 to EAB	LD	r, XL	1	1	1	0	1	0	1	0	1	0	r1	r0					5	r←XL
EB0 to EB3	LD	r, YP	1	1	1	0	1	0	1	1	0	0	r1	r0					5	r←YP
EB4 to EB7	LD	r, YH	1	1	1	0	1	0	1	1	0	1	r1	r0					5	r←YH
EB8 to EBB	LD	r, YL	1	1	1	0	1	0	1	1	1	0	r1	r0					5	r←YL
EC0 to ECF	LD	r, q	1	1	1	0	1	1	0	0	r1	r0	q1	q0					5	r←q
EE0	INC	X	1	1	1	0	1	1	1	0	0	0	0	0					5	X←X+1
EE0 to EEF	LDPX	r, q	1	1	1	0	1	1	1	0	r1	r0	q1	q0					5	r←q, X←X+1
EF0	INC	Y	1	1	1	0	1	1	1	1	0	0	0	0					5	Y←Y+1
EF0 to EFF	LDPY	r, q	1	1	1	0	1	1	1	1	r1	r0	q1	q0					5	r←q, Y←Y+1
F00 to F0F	CP	r, q	1	1	1	1	0	0	0	0	r1	r0	q1	q0			↕	↕	7	r-q
F10 to F1F	FAN	r, q	1	1	1	1	0	0	0	1	r1	r0	q1	q0			↕		7	r∧q
F28 to F2B	ACPX	MX, r	1	1	1	1	0	0	1	0	1	0	r1	r0		★	↕	↕	7	M(X)←M(X)+r+C, X←X+1
F2C to F2F	ACPY	MY, r	1	1	1	1	0	0	1	0	1	1	r1	r0		★	↕	↕	7	M(Y)←M(Y)+r+C, Y←Y+1
F38 to F3B	SCPX	MX, r	1	1	1	1	0	0	1	1	1	0	r1	r0		★	↕	↕	7	M(X)←M(X)-r-C, X←X+1
F3C to F3F	SCPY	MY, r	1	1	1	1	0	0	1	1	1	1	r1	r0		★	↕	↕	7	M(Y)←M(Y)-r-C, Y←Y+1
F40 to F4F	SET	F, i	1	1	1	1	0	1	0	0	i3	i2	i1	i0	↑	↑	↑	↑	7	F←F∨i3~i0
F41	SCF		1	1	1	1	0	1	0	0	0	0	0	1				↑	7	C←1
F42	SZF		1	1	1	1	0	1	0	0	0	0	1	0			↑		7	Z←1
F44	SDF		1	1	1	1	0	1	0	0	0	1	0	0		↑			7	D←1 (Decimal Adjuster ON)
F48	EI		1	1	1	1	0	1	0	0	1	0	0	0	↑				7	I←1 (Enables Interrupt)
F50 to F5F	RST	F, i	1	1	1	1	0	1	0	1	i3	i2	i1	i0	↓	↓	↓	↓	7	F←F∧i3~i0
F57	DI		1	1	1	1	0	1	0	1	0	1	1	1	↓				7	I←0 (Disables Interrupt)
F5B	RDF		1	1	1	1	0	1	0	1	1	0	1	1		↓			7	D←0 (Decimal Adjuster OFF)
F5D	RZF		1	1	1	1	0	1	0	1	1	1	0	1			↓		7	Z←0
F5E	RCF		1	1	1	1	0	1	0	1	1	1	1	0				↓	7	C←0
F60 to F6F	INC	Mn	1	1	1	1	0	1	1	0	n3	n2	n1	n0			↕	↕	7	M(n3~n0)←M(n3~n0)+1
F70 to F7F	DEC	Mn	1	1	1	1	0	1	1	1	n3	n2	n1	n0			↕	↕	7	M(n3~n0)←M(n3~n0)-1
F80 to F8F	LD	Mn, A	1	1	1	1	1	0	0	0	n3	n2	n1	n0					5	M(n3~n0)←A

Operation Code (HEX)	Mne-monic	Operand	Operation Code												Flag				Clock	Operation
			B	A	9	8	7	6	5	4	3	2	1	0	I	D	Z	C		
F90 to F9F	LD	Mn, B	1	1	1	1	1	0	0	1	n3	n2	n1	n0					5	M(n3~n0) ← B
FA0 to FAF	LD	A, Mn	1	1	1	1	1	0	1	0	n3	n2	n1	n0					5	A ← M(n3~n0)
FB0 to FBF	LD	B, Mn	1	1	1	1	1	0	1	1	n3	n2	n1	n0					5	B ← M(n3~n0)
FC0 to FC3	PUSH	r	1	1	1	1	1	1	0	0	0	0	r1	r0					5	SP ← SP-1, M(SP) ← r
FC4	PUSH	XP	1	1	1	1	1	1	0	0	0	1	0	0					5	SP ← SP-1, M(SP) ← XP
FC5	PUSH	XH	1	1	1	1	1	1	0	0	0	1	0	1					5	SP ← SP-1, M(SP) ← XH
FC6	PUSH	XL	1	1	1	1	1	1	0	0	0	1	1	0					5	SP ← SP-1, M(SP) ← XL
FC7	PUSH	YP	1	1	1	1	1	1	0	0	0	1	1	1					5	SP ← SP-1, M(SP) ← YP
FC8	PUSH	YH	1	1	1	1	1	1	0	0	1	0	0	0					5	SP ← SP-1, M(SP) ← YH
FC9	PUSH	YL	1	1	1	1	1	1	0	0	1	0	0	1					5	SP ← SP-1, M(SP) ← YL
FCA	PUSH	F	1	1	1	1	1	1	0	0	1	0	1	0					5	SP ← SP-1, M(SP) ← F
FCB	DEC	SP	1	1	1	1	1	1	0	0	1	0	1	1					5	SP ← SP-1
FD0 to FD3	POP	r	1	1	1	1	1	1	0	1	0	0	r1	r0					5	r ← M(SP), SP ← SP+1
FD4	POP	XP	1	1	1	1	1	1	0	1	0	1	0	0					5	XP ← M(SP), SP ← SP+1
FD5	POP	XH	1	1	1	1	1	1	0	1	0	1	0	1					5	XH ← M(SP), SP ← SP+1
FD6	POP	XL	1	1	1	1	1	1	0	1	0	1	1	0					5	XL ← M(SP), SP ← SP+1
FD7	POP	YP	1	1	1	1	1	1	0	1	0	1	1	1					5	YP ← M(SP), SP ← SP+1
FD8	POP	YH	1	1	1	1	1	1	0	1	1	0	0	0					5	YH ← M(SP), SP ← SP+1
FD9	POP	YL	1	1	1	1	1	1	0	1	1	0	0	1					5	YL ← M(SP), SP ← SP+1
FDA	POP	F	1	1	1	1	1	1	0	1	1	0	1	0	↕	↕	↕	↕	5	F ← M(SP), SP ← SP+1
FDB	INC	SP	1	1	1	1	1	1	0	1	1	0	1	1					5	SP ← SP+1
FDE	RETS		1	1	1	1	1	1	0	1	1	1	1	0					12	PCSL ← M(SP), PCSH ← M(SP+1), PCP ← M(SP+2) SP ← SP+3, PC ← PC+1
FDF	RET		1	1	1	1	1	1	0	1	1	1	1	1					7	PCSL ← M(SP), PCSH ← M(SP+1), PCP ← M(SP+2) SP ← SP+3
FE0 to FE3	LD	SPH, r	1	1	1	1	1	1	1	0	0	0	r1	r0					5	SPH ← r
FE4 to FE7	LD	r, SPH	1	1	1	1	1	1	1	0	0	1	r1	r0					5	r ← SPH
FE8	JPBA		1	1	1	1	1	1	1	0	1	0	0	0					5	PCB ← NBP, PCP ← NPP, PCSH ← B, PCSL ← A
FF0 to FF3	LD	SPL, r	1	1	1	1	1	1	1	1	0	0	r1	r0					5	SPL ← r
FF4 to FF7	LD	r, SPL	1	1	1	1	1	1	1	1	0	1	r1	r0					5	r ← SPL
FF8	HALT		1	1	1	1	1	1	1	1	1	0	0	0					5	Halt (stop clock)
FF9	SLP		1	1	1	1	1	1	1	1	1	0	0	1					5	SLEEP (stop oscillation)
FFB	NOP5		1	1	1	1	1	1	1	1	1	0	1	1					5	No operation (5 clock cycles)
FFF	NOP7		1	1	1	1	1	1	1	1	1	1	1	1					7	No operation (7 clock cycles)

3.2 Operands

This section describes the operands used in the instructions.

p 5-bit immediate data or labels 00H to 1FH. Used to specify a destination address.
s 8-bit immediate data or labels 00H to FFH. Used to specify a destination address.
e 8-bit immediate data 00H to FFH.
i 4-bit immediate data 00H to 0FH.
r 2-bit immediate data. See Table 3.2.1.
q 2-bit immediate data. See Table 3.2.1.

The contents of A, B, MX, MY are referenced using *r* and *q* as shown in the following table.

Table 3.2.1 Values of r and q

	*r*1 or *q*1	*r*0 or *q*0
A	0	0
B	0	1
MX	1	0
MY	1	1

A A register
B B register
XP XP register---four high-order bits of IX
YP YP register---four high-order bits of IY
X XHL register---eight low-order bits of IX
Y YHL register---eight low-order bits of IY
XH XH register---four high-order bits of XHL
XL XL register---four low-order bits of XHL
YH YH register---four high-order bits of YHL
YL YL register---four low-order bits of YHL
SP Stack pointer SP
SPH Four high-order bits of SP
SPL Four low-order bits of SP
F Flag register (IF, DF, ZF, CF)
MX Data memory location whose address is specified by IX
MY Data memory location whose address is specified by IY
M*n* Data memory location within the register area (000H to 00FH), specified by immediate data *n* (0H to FH)
C Carry
NC No carry
Z Zero
NZ Not zero

3.3 Flags

1. Carry flag

The carry flag is set if a carry was generated by the previous operation. It is affected by 17 arithmetic and logical instructions, four flag operations, eight index operation instructions and the POP F instruction.

2. Zero flag

The zero flag is set if a zero occurred in the previous operation. It is affected by 26 arithmetic and logical instructions, four flag operations, eight index operation instructions and the POP F instruction.

3. Decimal flag

The decimal flag enables decimal addition and subtraction when set. It is set by SDF or SET F,i and reset by RDF or RST F,*i*. It is affected by the POP F instruction.

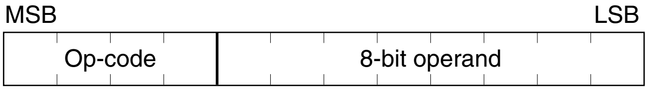
4. Interrupt flag

The interrupt flag enables interrupts when set. It is set by EI or SET F,*i* and reset by DI or RST F,*i*. It is affected by the POP F instruction. When an interrupt is generated, the I flag is automatically reset. It is not automatically set at the end of the interrupt service routine.

3.4 Instruction Types

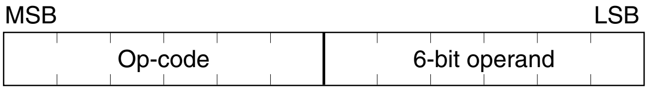
Instructions are divided into six types according to the size of the operand.

(I)



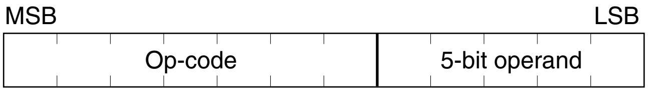
ex: JP s
CALL s
LBPX MX,e
etc.

(II)



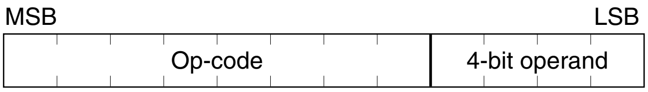
ex: ADD r, i
LD r, i
FAN r, i
etc.

(III)



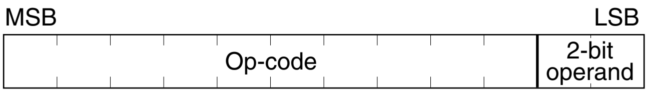
ex: PSET p

(IV)



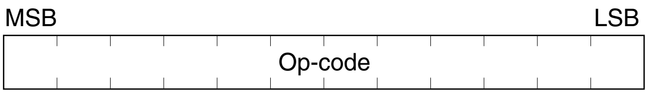
ex: SET F, i
LD r, q
INC Mn
etc.

(V)



ex: ACPX MX, r
LD XH, r
PUSH r
etc.

(VI)



ex: JPBA
POP YL
INC X
etc.

3.5 Instruction Descriptions

This section describes E0C6200/6200A instructions in alphabetical order.

ACPX MX,r *Add with carry r-register to M(X), increment X by 1*

Source Format: **ACPX MX,r**

Operation: M(X) ← M(X) + r + C, X ← X + 1

OP-Code:

1	1	1	1	0	0	1	0	1	0	r_1	r_0

MSB LSB F28H to F2BH

Type: V

Clock Cycles: 7

Flag:
- **C** – Set if a carry is generated; otherwise, reset.
- **Z** – Set if the result is zero; otherwise, reset.
- **D** – Not affected
- **I** – Not affected

Description: Adds the carry bit and the contents of the r-register to the data memory location addressed by IX. X is incremented by one. Incrementing X does not affect the flags.

Example:

	ACPX MX,A	ACPX MX,MY	
X register	1010 0000	1010 0001	1010 0010
Y register	0100 0110	0100 0110	0100 0110
Memory (A0H)	0110	1111	1111
Memory (A1H)	0011	0011	0111
Memory (46H)	0100	0100	0100
A register	1000	1000	1000
C flag	1	0	0
Z flag	0	0	0

ACPY MY,r *Add with carry r-register to M(Y), increment Y by 1*

Source Format: **ACPY MY,r**

Operation: M(Y) ← M(Y) + r + C, Y ← Y + 1

OP-Code:

1	1	1	1	0	0	1	0	1	1	r_1	r_0

MSB LSB F2CH to F2FH

Type: V

Clock Cycles: 7

Flag:
- **C** – Set if a carry is generated; otherwise, reset.
- **Z** – Set if the result is zero; otherwise, reset.
- **D** – Not affected
- **I** – Not affected

Description: Adds the carry bit and the contents of the r-register to the data memory location addressed by IY. Y is incremented by one. Incrementing Y does not affect the flags.

Example:

	ACPY MY,A	ACPY MY,MX	
X register	0010 0001	0010 0001	0010 0001
Y register	0000 1110	0000 1111	0001 0000
Memory (0EH)	1000	1011	1011
Memory (0FH)	0100	0100	1010
Memory (21H)	0110	0110	0110
A register	0010	0010	0010
C flag	1	0	0
Z flag	0	0	0

ADC r,i

Add with carry immediate data i to r-register

Source Format: **ADC r,i**

Operation: $r \leftarrow r + i_3$ to $i_0 + C$

OP-Code:

1	1	0	0	0	1	r_1	r_0	i_3	i_2	i_1	i_0

MSB … LSB — C40H to C7FH

Type: II

Clock Cycles: 7

Flag:

C – Set if a carry is generated; otherwise, reset.
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Adds the carry bit and immediate data i to the r-register.

Example:

	ADC MX,3		ADC B,7
Memory (MX)	0100	1000	1000
B register	1001	1001	0000
C flag	1	0	1
Z flag	1	0	1

ADC r,q

Add with carry q-register to r-register

Source Format: **ADC r,q**

Operation: $r \leftarrow r + q + C$

OP-Code:

1	0	1	0	1	0	0	1	r_1	r_0	q_1	q_0

MSB … LSB — A90H to A9FH

Type: IV

Clock Cycles: 7

Flag:

C – Set if a carry is generated; otherwise, reset.
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Adds the carry bit and the contents of the q-register to the r-register.

Example:

	ADC MY,A		ADC MX,B
A register	0101	0101	0101
B register	0001	0001	0001
Memory (MX)	0111	0111	1001
Memory (MY)	1011	0001	0001
C flag	1	1	0
Z flag	0	0	0

ADC XH,i — *Add with carry immediate data i to XH*

Source Format: **ADC XH,i**

Operation: XH ← XH + i_3 to i_0 + C

OP-Code:

1	0	1	0	0	0	0	0	i_3	i_2	i_1	i_0

MSB ... LSB — A00H to A0FH

Type: IV

Clock Cycles: 7

Flag:
- **C** – Set if a carry is generated; otherwise, reset.
- **Z** – Set if the result is zero; otherwise, reset.
- **D** – Not affected
- **I** – Not affected

Description: Adds the carry bit and immediate data i to XH, the four high-order bits of XHL.

Example:

	ADC XH,2		ADC XH,4
XH register	1001	1100	0000
C flag	1	0	1
Z flag	0	0	1

ADC XL,i — *Add with carry immediate data i to XL*

Source Format: **ADC XL,i**

Operation: XL ← XL + i_3 to i_0 + C

OP-Code:

1	0	1	0	0	0	0	1	i_3	i_2	i_1	i_0

MSB ... LSB — A10H to A1FH

Type: IV

Clock Cycles: 7

Flag:
- **C** – Set if a carry is generated; otherwise, reset.
- **Z** – Set if the result is zero; otherwise, reset.
- **D** – Not affected
- **I** – Not affected

Description: Adds the carry bit and immediate data i to XL, the four low-order bits of XHL.

Example:

	ADC XL,3		ADC XL,0EH
XL register	0000	0100	0010
C flag	1	0	1
Z flag	1	0	0

ADC YH,i — *Add with carry immediate data i to YH*

Source Format: **ADC YH,i**

Operation: YH ← YH + i_3 to i_0 + C

OP-Code:

1	0	1	0	0	0	1	0	i_3	i_2	i_1	i_0

MSB … LSB — A20H to A2FH

Type: IV

Clock Cycles: 7

Flag:
C – Set if a carry is generated; otherwise, reset.
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Adds the carry bit and immediate data i to YH, the four high-order bits of YHL.

Example:

	ADC YH,3		ADC YH,6
YH register	1010	1110	0100
C flag	1	0	1
Z flag	0	0	0

ADC YL,i — *Add with carry immediate data i to YL*

Source Format: **ADC YL,i**

Operation: YL ← YL + i_3 to i_0 + C

OP-Code:

1	0	1	0	0	0	1	1	i_3	i_2	i_1	i_0

MSB … LSB — A30H to A3FH

Type: IV

Clock Cycles: 7

Flag:
C – Set if a carry is generated; otherwise, reset.
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Adds the carry bit and immediate data i to YL, the four low-order bits of YHL.

Example:

	ADC YL,3		ADC YL,2
YL register	1010	1110	0000
C flag	1	0	1
Z flag	0	0	1

ADD r,i — *Add immediate data i to r-register*

Source Format: **ADD r,i**

Operation: $r \leftarrow r + i_3$ to i_0

OP-Code:

1	1	0	0	0	0	r_1	r_0	i_3	i_2	i_1	i_0

MSB ... LSB — C00H to C3FH

Type: II

Clock Cycles: 7

Flag:
C – Set if a carry is generated; otherwise, reset.
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Adds immediate data i to the contents of the r-register.

Example:

	ADD A,5		ADD MY,2
A register	1010	1111	1111
Memory (MY)	0110	0110	1000
C flag	1	0	0
Z flag	0	0	0

ADD r,q — *Add q-register to r-register*

Source Format: **ADD r,q**

Operation: $r \leftarrow r + q$

OP-Code:

1	0	1	0	1	0	0	0	r_1	r_0	q_1	q_0

MSB ... LSB — A80H to A8FH

Type: IV

Clock Cycles: 7

Flag:
C – Set if a carry is generated; otherwise, reset.
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Adds the contents of the q-register to the contents of the r-register.

Example:

	ADD A,MY		ADD MX,B
A register	0010	1111	1111
B register	0100	0100	0100
Memory (MX)	0111	0111	1011
Memory (MY)	1101	1101	1101
C flag	1	0	0
Z flag	1	0	0

AND r,i — *Logical AND immediate data i with r-register*

Source Format: **AND r,i**

Operation: $r \leftarrow r \wedge i_3$ to i_0

OP-Code:

1	1	0	0	1	0	r_1	r_0	i_3	i_2	i_1	i_0

MSB … LSB   C80H to CBFH

Type: II

Clock Cycles: 7

Flag:
C – Not affected
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Performs a logical AND operation between immediate data i and the contents of the r-register. The result is stored in the r-register.

Example:

	AND A,5		AND MX,3
A register	0110	0100	0100
Memory (MX)	1000	1000	0000
C flag	1	1	1
Z flag	0	0	1

AND r,q — *Logical AND q-register with r-register*

Source Format: **AND r,q**

Operation: $r \leftarrow r \wedge q$

OP-Code:

1	0	1	0	1	1	0	0	r_1	r_0	q_1	q_0

MSB … LSB   AC0H to ACFH

Type: IV

Clock Cycles: 7

Flag:
C – Not affected
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Performs a logical AND operation between the contents of the q-register and the contents of the r-register. The result is stored in the r-register.

Example:

	AND MX,A		AND B,MY
A register	0100	0100	0100
B register	1011	1011	0010
Memory (MX)	1010	0000	0000
Memory (MY)	0010	0010	0010
C flag	0	0	0
Z flag	0	1	0

CALL s — *Call subroutine*

Source Format: **CALL s**

Operation: M(SP-1) ← PCP, M(SP-2) ← PCSH, M(SP-3) ← PCSL + 1, SP ← SP - 3,
PCP ← NPP, PCS ← s_7 to s_0

OP-Code:

0	1	0	0	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0

MSB ... LSB   400H to 4FFH

Type: I

Clock Cycles: 7

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Pushes the program counter (PCP, PCS) onto the stack as the return address, then calls the subroutine addressed by NPP and the 8-bit operand.

Example:

	PSET 06H		CALL 10H
PCP	0011	0011	0110
PCS	0010 1100	0010 1100	0001 0000
NPP	0001	0110	0110
SP	C0	C0	BD
Memory (SP-1)	xxxx	xxxx	0011
Memory (SP-2)	xxxx	xxxx	0010
Memory (SP-3)	xxxx	xxxx	1101

CALZ s — *Call subroutine at page zero*

Source Format: **CALZ s**

Operation: M(SP-1) ← PCP, M(SP-2) ← PCSH, M(SP-3) ← PCSL + 1, SP ← SP - 3,
PCP ← 0, PCS ← s_7 to s_0

OP-Code:

0	1	0	1	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0

MSB ... LSB   500H to 5FFH

Type: I

Clock Cycles: 7

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Pushes the program counter (PCP, PCS) onto the stack as the return address, then calls the subroutine addressed by the 8-bit operand. As NPP is reset to 0H, only a subroutine in page 0 can be called.

Example:

	CALZ 10H	
PCP	1010	0000
PCS	0010 1110	0001 0000
SP	CA	C7
Memory (SP-1)	xxxx	1010
Memory (SP-2)	xxxx	0010
Memory (SP-3)	xxxx	1111

CP r,i — *Compare immediate data i with r-register*

Source Format: **CP r,i**

Operation: r - i_3 to i_0

OP-Code:

1	1	0	1	1	1	r_1	r_0	i_3	i_2	i_1	i_0

MSB … LSB — DC0H to DFFH

Type: II

Clock Cycles: 7

Flag:
C – Set if r < i_3 to i_0; otherwise, reset.
Z – Set if r = i_3 to i_0; otherwise, reset.
D – Not affected
I – Not affected

Description: Compares immediate data i to the r-register by subtracting i from the contents of r. The r-register remains unchanged.

1. When Z = 0 and C = 0 then i < r
2. When Z = 1 and C = 0 then i = r
3. When Z = 0 and C = 1 then i > r

Example:

		CP A,4	CP MX,7	CP B,2
A register	0100	0100	0100	0100
B register	1010	1010	1010	1010
Memory (MX)	0010	0010	0010	0010
C flag	1	0	1	0
Z flag	0	1	0	0

CP r,q — *Compare q-register with r-register*

Source Format: **CP r,q**

Operation: r - q

OP-Code:

1	1	1	1	0	0	0	0	r_1	r_0	q_1	q_0

MSB … LSB — F00H to F0FH

Type: IV

Clock Cycles: 7

Flag:
C – Set if r < q; otherwise, reset.
Z – Set if r = q; otherwise, reset.
D – Not affected
I – Not affected

Description: Compares the q-register to the r-register by subtracting the contents of q from the contents of r. The registers remain unchanged.

1. When Z = 0 and C = 0 then q < r
2. When Z = 1 and C = 0 then q = r
3. When Z = 0 and C = 1 then q > r

Example:

		CP A,B	CP MY,A
A register	1000	1000	1000
B register	0100	0100	0100
Memory (MY)	0111	0111	0111
C flag	0	0	1
Z flag	0	0	0

CP XH,i — *Compare immediate data i with XH*

Source Format: **CP XH,i**

Operation: XH - i_3 to i_0

OP-Code:

1	0	1	0	0	1	0	0	i_3	i_2	i_1	i_0

MSB ... LSB — A40H to A4FH

Type: IV

Clock Cycles: 7

Flag:
C – Set if XH < i_3 to i_0; otherwise, reset.
Z – Set if XH = i_3 to i_0; otherwise, reset.
D – Not affected
I – Not affected

Description: Compares immediate data i to XH by subtracting i from the contents of XH. XH remains unchanged.

1. When Z = 0 and C = 0 then i < XH
2. When Z = 1 and C = 0 then i = XH
3. When Z = 0 and C = 1 then i > XH

Example:

		CP XH,2	CP XH,4	CP XH,9
XH register	0100	0100	0100	0100
C flag	1	0	0	1
Z flag	0	0	1	0

CP XL,i — *Compare immediate data i with XL*

Source Format: **CP XL,i**

Operation: XL - i_3 to i_0

OP-Code:

1	0	1	0	0	1	0	1	i_3	i_2	i_1	i_0

MSB ... LSB — A50H to A5FH

Type: IV

Clock Cycles: 7

Flag:
C – Set if XL < i_3 to i_0; otherwise, reset.
Z – Set if XL = i_3 to i_0; otherwise, reset.
D – Not affected
I – Not affected

Description: Compares immediate data i to XL by subtracting i from the contents of XL. XL remains unchanged.

1. When Z = 0 and C = 0 then i < XL
2. When Z = 1 and C = 0 then i = XL
3. When Z = 0 and C = 1 then i > XL

Example:

		CP XL,7	CP XL,9	CP XL,0AH
XL register	1001	1001	1001	1001
C flag	0	0	0	1
Z flag	0	0	1	0

CP YH,i

Compare immediate data i with YH

Source Format: **CP YH,i**

Operation: YH - i_3 to i_0

OP-Code:

1	0	1	0	0	1	1	0	i_3	i_2	i_1	i_0

MSB … LSB — A60H to A6FH

Type: IV

Clock Cycles: 7

Flag:
C – Set if YH < i_3 to i_0; otherwise, reset.
Z – Set if YH = i_3 to i_0; otherwise, reset.
D – Not affected
I – Not affected

Description: Compares immediate data i to YH by subtracting i from the contents of YH. YH remains unchanged.

1. When Z = 0 and C = 0 then i < YH
2. When Z = 1 and C = 0 then i = YH
3. When Z = 0 and C = 1 then i > YH

Example:

	CP YH,0AH		CP YH,3	CP YH,0FH
YH register	1010	1010	1010	1010
C flag	1	0	0	1
Z flag	0	1	0	0

CP YL,i

Compare immediate data i with YL

Source Format: **CP YL,i**

Operation: YL - i_3 to i_0

OP-Code:

1	0	1	0	0	1	1	1	i_3	i_2	i_1	i_0

MSB … LSB — A70H to A7FH

Type: IV

Clock Cycles: 7

Flag:
C – Set if YL < i_3 to i_0; otherwise, reset.
Z – Set if YL = i_3 to i_0; otherwise, reset.
D – Not affected
I – Not affected

Description: Compares immediate data i to YL by subtracting i from the contents of YL. YL remains unchanged.

1. When Z = 0 and C = 0 then i < YL
2. When Z = 1 and C = 0 then i = YL
3. When Z = 0 and C = 1 then i > YL

Example:

	CP YL,5		CP YL,1	CP YL,4
YL register	0100	0100	0100	0100
C flag	0	1	0	0
Z flag	1	0	0	1

DEC Mn — *Decrement memory*

Source Format: **DEC Mn**

Operation: M(n_3 to n_0) ← M(n_3 to n_0) - 1

OP-Code:

MSB											LSB
1	1	1	1	0	1	1	1	n_3	n_2	n_1	n_0

F70H to F7FH

Type: IV

Clock Cycles: 7

Flag:
C – Set if a borrow is generated; otherwise, reset.
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Decrements the contents of the data memory location addressed by Mn by 1.

Example:

	DEC M0		DEC M2	DEC M0FH
Memory (00H)	1001	1000	1000	1000
Memory (02H)	0000	0000	1111	1111
Memory (0FH)	0001	0001	0001	0000
C flag	1	0	1	0
Z flag	0	0	0	1

DEC SP — *Decrement stack pointer*

Source Format: **DEC SP**

Operation: SP ← SP - 1

OP-Code:

MSB											LSB
1	1	1	1	1	1	0	0	1	0	1	1

FCBH

Type: VI

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Decrements the contents of the stack pointer by 1. This operation does not affect the flags.

Example:

	DEC SP	
Memory (SP)	1011 0001	1011 0000
C flag	0	0
Z flag	1	1

DI

Disable interrupts

Source Format: **DI**

Operation: I ← 0

OP-Code:

1	1	1	1	0	1	0	1	0	1	1	1

MSB LSB

F57H

Type: VI

Clock Cycles: 7

Flag:

C – Not affected
Z – Not affected
D – Not affected
I – Reset

Description: Disables all interrupts.

Example:

	DI	
C flag	0	0
Z flag	1	1
D flag	0	0
I flag	1	0

EI

Enable interrupts

Source Format: **EI**

Operation: I ← 1

OP-Code:

1	1	1	1	0	1	0	0	1	0	0	0

MSB LSB

F48H

Type: VI

Clock Cycles: 7

Flag:

C – Not affected
Z – Not affected
D – Not affected
I – Set

Description: Enables all interrupts.

Example:

	EI	
C flag	1	1
Z flag	0	0
D flag	0	0
I flag	0	1

FAN r,i — *Logical AND immediate data i with r-register for flag check*

Source Format: **FAN r,i**

Operation: r ∧ i_3 to i_0

OP-Code:

1	1	0	1	1	0	r_1	r_0	i_3	i_2	i_1	i_0

MSB … LSB — D80H to DBFH

Type: II

Clock Cycles: 7

Flag:
C – Not affected
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Performs a logical AND operation between immediate data i and the contents of the r-register. Only the Z flag is affected. The r-register remains unchanged.

Example:

		FAN A,7	FAN MY,9	FAN B,2
A register	1000	1000	1000	1000
B register	0100	0100	0100	0100
Memory (MY)	1000	1000	1000	1000
C flag	1	1	1	1
Z flag	0	1	0	1

FAN r,q — *Logical AND q-register with r-register for flag check*

Source Format: **FAN r,q**

Operation: r ∧ q

OP-Code:

1	1	1	1	0	0	0	1	r_1	r_0	q_1	q_0

MSB … LSB — F10H to F1FH

Type: IV

Clock Cycles: 7

Flag:
C – Not affected
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Performs a logical AND operation between the contents of the q-register and the contents of the r-register. Only the Z flag is affected. The registers remains unchanged.

Example:

		FAN A,B	FAN MX,B	FAN A,MY
A register	1000	1000	1000	1000
B register	1010	1010	1010	1010
Memory (MX)	0101	0101	0101	0101
Memory (MY)	1110	1110	1110	1110
C flag	0	0	0	0
Z flag	0	0	1	0

HALT *Halt*

Source Format: **HALT**

Operation: Stops CPU

OP-Code:

1	1	1	1	1	1	1	1	1	0	0	0

MSB ... LSB — FF8H

Type: VI

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Stops the CPU. When an interrupt occurs, PCP and PCS are pushed onto the stack as the return address and the interrupt service routine is executed.

Example:

	Instruction	State	PCP	PCS	I flag
	HALT	RUN	0001	0011 0011	1
		HALT			
Interrupt			0001	0011 0100	1
		RUN	0001	Interrupt vector address	0

INC Mn *Increment memory by 1*

Source Format: **INC Mn**

Operation: $M(n_3 \text{ to } n_0) \leftarrow M(n_3 \text{ to } n_0) + 1$

OP-Code:

1	1	1	1	0	1	1	0	n_3	n_2	n_1	n_0

MSB ... LSB — F60H to F6FH

Type: IV

Clock Cycles: 7

Flag:
C – Set if a carry is generated; otherwise, reset.
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: The contents of the data memory location addressed by Mn is incremented by 1.

Example:

		INC M1	INC M3	INC M0DH
Memory (01H)	0100	0101	0101	0101
Memory (03H)	1111	1111	0000	0000
Memory (0DH)	0111	0111	0111	1000
C flag	0	0	1	0
Z flag	1	0	1	0

INC SP — *Increment stack pointer by 1*

Source Format: **INC SP**

Operation: SP ← SP + 1

OP-Code:

1	1	1	1	1	1	0	1	1	0	1	1

MSB … LSB FDBH

Type: VI

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Increments the contents of the stack pointer by 1. This operation does not affect the flags.

Example:

	INC SP	
SP	1110 1111	1111 0000
C flag	0	0
Z flag	0	0

INC X — *Increment X-register by 1*

Source Format: **INC X**

Operation: X ← X + 1

OP-Code:

1	1	1	0	1	1	1	0	0	0	0	0

MSB … LSB EE0H

Type: VI

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Increments the contents of register X by 1. This operation does not affect the flags.

Example:

	INC X	
X register	1111 1110	1111 1111
C flag	1	1
Z flag	0	0

INC Y — *Increment Y-register by 1*

Source Format: **INC Y**

Operation: Y ← Y + 1

OP-Code:

1	1	1	0	1	1	1	1	0	0	0	0

MSB … LSB   EF0H

Type: VI

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Increments the contents of register Y by 1. This operation does not affect the flags.

Example:

	INC Y	
Y register	1011 0111	1011 1000
C flag	1	1
Z flag	0	0

JPBA — *Indirect jump using registers A and B*

Source Format: **JPBA**

Operation: PCB ← NBP, PCP ← NPP, PCSH ← B, PCSL ← A

OP-Code:

1	1	1	1	1	1	1	0	1	0	0	0

MSB … LSB   FE8H

Type: VI

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Uses the contents of a- and b-registers to specify the destination address of the jump. The b-register contains the four high-order bits of the address and the a-register contains the four low-order bits of the address.

Example:

	PSET 15H		JPBA
PCB	0	0	1
NBP	0	1	1
PCP	1000	1000	0101
NPP	0001	0101	0101
PCS	1001 0000	1001 0001	0000 0110
A register	0110	0110	0110
B register	0000	0000	0000

JP C,s *Jump if carry flag is set*

Source Format: **JP C,s**

Operation: PCB ← NBP, PCP ← NPP, PCS ← s_7 to s_0 if C = 1

OP-Code:

0	0	1	0	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0

MSB ... LSB — 200H to 2FFH

Type: I

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Jumps to the destination address specified by the 8-bit operand when the carry flag is set.

Example:

	ADD A,8		PSET 06H	JP C,10H
PCB	0	0	0	0
NBP	0	0	0	0
PCP	0010	0010	0010	0110
NPP	0001	0001	0110	0110
PCS	0011 1100	0011 1101	0011 1110	0001 0000
A register	1000	0000	0000	0000
C flag	0	1	1	1

JP NC,s *Jump if not carry*

Source Format: **JP NC,s**

Operation: PCB ← NBP, PCP ← NPP, PCS ← s_7 to s_0 if C = 0

OP-Code:

0	0	1	1	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0

MSB ... LSB — 300H to 3FFH

Type: I

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Jumps to the destination address specified by the 8-bit operand when the carry flag is not set.

Example:

	PSET 11H		JP NC,10H
PCB	0	0	1
NBP	0	1	1
PCP	1001	1001	0001
NPP	0001	0001	0001
PCS	1000 1111	1001 0000	0001 0000
C flag	0	0	0

JP NZ,s

Jump if not zero

Source Format: **JP NZ,s**

Operation: PCB ← NBP, PCP ← NPP, PCS ← s_7 to s_0 if Z = 0

OP-Code:

0	1	1	1	s7	s6	s5	s4	s3	s2	s1	s0

MSB … LSB — 700H to 7FFH

Type: I

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Jumps to the destination address specified by the 8-bit operand when the zero flag is not set.

Example:

	JP NZ,10H	
PCB	1	1
NBP	1	1
PCP	0000	0000
NPP	0000	0000
PCS	0000 0111	0001 0000
Z flag	0	0

JP s

Jump

Source Format: **JP s**

Operation: PCB ← NBP, PCP ← NPP, PCS ← s_7 to s_0

OP-Code:

0	0	0	0	s7	s6	s5	s4	s3	s2	s1	s0

MSB … LSB — 000H to 0FFH

Type: I

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Unconditional jump to the destination address specified by the 8-bit operand.

Example:

	PSET 0AH		JP 10H
PCB	0	0	0
NBP	0	0	0
PCP	0000	0000	1010
NPP	0001	1010	1010
PCS	0100 0010	0100 0011	0001 0000

JP Z,s — *Jump if zero*

Source Format: **JP Z,s**

Operation: PCB ← NBP, PCP ← NPP, PCS ← s_7 to s_0 if Z = 1

OP-Code:

0	1	1	0	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0

MSB … LSB   600H to 6FFH

Type: I

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Jumps to the destination address specified by the 8-bit operand when the zero flag is set.

Example:

		SUB A,B	PSET 1BH	JP Z,10H
PCB	0	0	0	1
NBP	0	0	1	1
PCP	0101	0101	0101	1011
NPP	0001	0001	1011	1011
PCS	0000 0010	0000 0011	0000 0100	0001 0000
A register	0110	0000	0000	0000
B register	0110	0110	0110	0110
Z flag	0	1	1	1

LBPX MX,e — *Load immediate data e to memory, and increment X by 2*

Source Format: **LBPX MX,e**

Operation: M(X) ← e_3 to e_0, M(X+1) ← e_7 to e_4, X ← X + 2

OP-Code:

1	0	0	1	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0

MSB … LSB   900H to 9FFH

Type: I

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Stores 8-bit immediate data e in two, consecutive 4-bit locations in data memory. The X-register is incremented by 2. An overflow in X does not affect the flags.

Example:

		LBPX MX,18H	LBPX MX,36H
X register	0001 1110	0010 0000	0010 0010
Memory (1EH)	0010	1000	1000
Memory (1FH)	1111	0001	0001
Memory (20H)	0000	0000	0110
Memory (21H)	0111	0111	0011

LD A,Mn *Load memory into A-register*

Source Format: **LD A,Mn**

Operation: A ← M(n_3 to n_0)

OP-Code:

1	1	1	1	1	0	1	0	n_3	n_2	n_1	n_0

MSB ... LSB — FA0H to FAFH

Type: IV

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the data memory location addressed by Mn into the A-register.

Example:

	LD A,M5		**LD A,M6**
A register	0100	1111	0100
Memory (05H)	1111	1111	1111
Memory (06H)	0100	0100	0100

LD B,Mn *Load memory into B-register*

Source Format: **LD B,Mn**

Operation: B ← M(n_3 to n_0)

OP-Code:

1	1	1	1	1	0	1	1	n_3	n_2	n_1	n_0

MSB ... LSB — FB0H to FBFH

Type: IV

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the data memory location addressed by Mn into the B-register.

Example:

	LD B,M7		**LD B,M8**
B register	0100	0110	1010
Memory (07H)	0110	0110	0110
Memory (08H)	1010	1010	1010

LD Mn,A — *Load A-register into memory*

Source Format: **LD Mn,A**

Operation: M(n_3 to n_0) ← A

OP-Code:

1	1	1	1	1	0	0	0	n_3	n_2	n_1	n_0

MSB … LSB — F80H to F8FH

Type: IV

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the A-register into the location addressed by Mn.

Example:

	LD M0AH,A		LD M0BH,A
A register	0110	0110	0110
Memory (0AH)	0100	0110	0110
Memory (0BH)	1011	1011	0110

LD Mn,B — *Load B-register into memory*

Source Format: **LD Mn,B**

Operation: M(n_3 to n_0) ← B

OP-Code:

1	1	1	1	1	0	0	1	n_3	n_2	n_1	n_0

MSB … LSB — F90H to F9FH

Type: IV

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the B-register into the data memory location addressed by Mn.

Example:

	LD M0,B		LD M1,B
B register	0100	0100	0100
Memory (00H)	1011	0100	0100
Memory (01H)	1111	1111	0100

LDPX MX,i *Load immediate data i into MX, increment X by 1*

Source Format: **LDPX MX,i**

Operation: M(X) ← i_3 to i_0, X ← X + 1

OP-Code:

1	1	1	0	0	1	1	0	i_3	i_2	i_1	i_0

MSB … LSB E60H to E6FH

Type: IV

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads immediate data i into the data memory location addressed by IX. X is incremented by 1. Incrementing X does not affect the flags.

Example:

	LDPX MX,7		LDPX MX,0AH
X register	1000 0011	1000 0100	1000 0101
Memory (83H)	0010	0111	0111
Memory (84H)	1001	1001	1010

LDPX r,q *Load q-register into r-register, increment X by 1*

Source Format: **LDPX r,q**

Operation: r ← q, X ← X + 1

OP-Code:

1	1	1	0	1	1	1	0	r_1	r_0	q_1	q_0

MSB … LSB EE0H to EEFH

Type: IV

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the q-register into the r-register. X is incremented by 1. Incrementing X does not affect the flags.

Example:

	LDPX A,B		LDPX B,MY
X register	0100 1001	0100 1010	0100 1011
A register	1010	1101	1101
B register	1101	1101	0000
Memory (MY)	0000	0000	0000

LDPY MY,i *Load immediate data i into MY, increment Y by 1*

Source Format: **LDPY MY,i**

Operation: M(Y) ← i_3 to i_0, Y ← Y + 1

OP-Code:

1	1	1	0	0	1	1	1	i_3	i_2	i_1	i_0

MSB ... LSB — E70H to E7FH

Type: IV

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads immediate data i into the data memory location addressed by IY. Y is incremented by 1. Incrementing Y does not affect the flags.

Example:

	LDPY MY,7		LDPY MY,0
Y register	0010 1101	0010 1110	0010 1111
Memory (2DH)	1010	0111	0111
Memory (2EH)	0010	0010	0000

LDPY r,q *Load q-register into r-register, increment Y by 1*

Source Format: **LDPY r,q**

Operation: r ← q, Y ← Y + 1

OP-Code:

1	1	1	0	1	1	1	1	r_1	r_0	q_1	q_0

MSB ... LSB — EF0H to EFFH

Type: IV

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the q-register into the r-register. Y is incremented by 1. Incrementing Y does not affect the flags.

Example:

	LDPY A,B		LDPY MX,B
Y register	0100 1000	0100 1001	0100 1010
A register	1010	1000	1000
B register	1000	1000	1000
Memory (MX)	0010	0010	1000

LD r,i

Load immediate data i into r-register

Source Format: **LD r,i**

Operation: r ← i_3 to i_0

OP-Code:

1	1	1	0	0	0	r_1	r_0	i_3	i_2	i_1	i_0

MSB ... LSB — E00H to E3FH

Type: II

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads immediate data i into the r-register.

Example:

	LD A,6		LD MY,0
A register	0101	0110	0110
Memory (MY)	1001	1001	0000

LD r,q

Load q-register into r-register

Source Format: **LD r,q**

Operation: r ← q

OP-Code:

1	1	1	0	1	1	0	0	r_1	r_0	q_1	q_0

MSB ... LSB — EC0H to ECFH

Type: IV

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: The contents of the q-register are loaded into the r-register.

Example:

	LD A,B		LD B,MY
A register	0010	0000	0000
B register	0000	0000	0110
Memory (MY)	0110	0110	0110

LD r,SPH *Load SPH into r-register*

Source Format: **LD r,SPH**

Operation: r ← SPH

OP-Code:

1	1	1	1	1	1	1	0	0	1	r_1	r_0

MSB LSB

FE4H to FE7H

Type: V

Clock Cycles: 5

Flag: **C** – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the four high-order bits of the stack pointer into the r-register.

Example:

	LD MX,SPH		LD A,SPH
SPH	0111	0111	0111
A register	0000	0000	0111
Memory (MX)	1100	0111	0111

LD r,SPL *Load SPL into r-register*

Source Format: **LD r,SPL**

Operation: r ← SPL

OP-Code:

1	1	1	1	1	1	1	1	0	1	r_1	r_0

MSB LSB

FF4H to FF7H

Type: V

Clock Cycles: 5

Flag: **C** – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the four low-order bits of the stack pointer into the r-register.

Example:

	LD A,SPL		LD MY,SPL
SPL	1001	1001	1001
A register	0010	1001	1001
Memory (MY)	0000	0000	1001

LD r,XH *Load XH into r-register*

Source Format: **LD r,XH**

Operation: r ← XH

OP-Code:

1	1	1	0	1	0	1	0	0	1	r_1	r_0

MSB LSB

EA4H to EA7H

Type: V

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the four high-order bits of register X into the r-register.

Example:

	LD B,XH		LD MX,XH
XH register	1010	1010	1010
B register	0010	1010	1010
Memory (MX)	0000	0000	1010

LD r,XL *Load XL into r-register*

Source Format: **LD r,XL**

Operation: r ← XL

OP-Code:

1	1	1	0	1	0	1	0	1	0	r_1	r_0

MSB LSB

EA8H to EABH

Type: V

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the four low-order bits of register X into the r-register.

Example:

	LD MY,XL		LD A,XL
XL register	0000	0000	0000
A register	1101	1101	0000
Memory (MY)	0001	0000	0000

LD r,XP — *Load XP into r-register*

Source Format: **LD r,XP**

Operation: r ← XP

OP-Code:

1	1	1	0	1	0	1	0	0	0	r_1	r_0

MSB … LSB — EA0H to EA3H

Type: V

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the 4-bit page part of index register IX into the r-register.

Example:

	LD MX,XP		LD A,XP
XP register	1111	1111	1111
A register	0010	0010	1111
Memory (MX)	0101	1111	1111

LD r,YH — *Load YH into r-register*

Source Format: **LD r,YH**

Operation: r ← YH

OP-Code:

1	1	1	0	1	0	1	1	0	1	r_1	r_0

MSB … LSB — EB4H to EB7H

Type: V

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the four high-order bits of register Y into the r-register.

Example:

	LD A,YH		LD MY,YH
YH register	1010	1010	1010
A register	1100	1010	1010
Memory (MY)	1110	1110	1010

LD r,YL

Load YL into r-register

Source Format: **LD r,YL**

Operation: r ← YL

OP-Code:

1	1	1	0	1	0	1	1	1	0	r_1	r_0

MSB LSB

EB8H to EBBH

Type: V

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the four low-order bits of register Y into the r-register.

Example:

	LD B,YL		LD MX,YL
YL register	0000	0000	0000
B register	0110	0000	0000
Memory (MX)	1011	1011	0000

LD r,YP

Load YP into r-register

Source Format: **LD r,YP**

Operation: r ← YP

OP-Code:

1	1	1	0	1	0	1	1	0	0	r_1	r_0

MSB LSB

EB0H to EB3H

Type: V

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the 4-bit page part of index register IY into the r-register.

Example:

	LD MY,YP		LD B,YP
YP register	1010	1010	1010
B register	1100	1100	1010
Memory (MY)	0110	1010	1010

LD SPH,r *Load r-register into SPH*

Source Format: **LD SPH,r**

Operation: SPH ← r

OP-Code:

1	1	1	1	1	1	1	0	0	0	r_1	r_0

MSB LSB

FE0H to FE3H

Type: V

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the r-register into the four high-order bits of the stack pointer.

Example:

	LD SPH,A		LD SPH,MY
SPH	1001	0011	1100
A register	0011	0011	0011
Memory (MY)	1100	1100	1100

LD SPL,r *Load r-register into SPL*

Source Format: **LD SPL,r**

Operation: SPL ← r

OP-Code:

1	1	1	1	1	1	1	1	0	0	r_1	r_0

MSB LSB

FF0H to FF3H

Type: V

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the r-register into the four low-order bits of the stack pointer.

Example:

	LD SPL,B		LD SPL,MX
SPL	1011	0111	1111
B register	0111	0111	0111
Memory (MX)	1111	1111	1111

LD X,e

Load immediate data e into X-register

Source Format: **LD X,e**

Operation: XH ← e_7 to e_4, XL ← e_3 to e_0

OP-Code:

1	0	1	1	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0

MSB ... LSB

B00H to BFFH

Type: I

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads 8-bit immediate data e into register X.

Example:

	LD X,6FH	
XH register	0000	0110
XL register	1011	1111

LD XH,r

Load r-register into XH

Source Format: **LD XH,r**

Operation: XH ← r

OP-Code:

1	1	1	0	1	0	0	0	0	1	r_1	r_0

MSB ... LSB

E84H to E87H

Type: V

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the r-register into the four high-order bits of register X.

Example:

	LD XH,A		LD XH,MY
XH register	0000	1011	0110
A register	1011	1011	1011
Memory (MY)	0110	0110	0110

LD XL,r *Load r-register into XL*

Source Format: **LD XL,r**

Operation: XL ← r

OP-Code:

1	1	1	0	1	0	0	0	1	0	r_1	r_0

MSB LSB

E88H to E8BH

Type: V

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the r-register into the four low-order bits of register X.

Example:

	LD XL,MY		LD XL,A
XL register	0000	0010	1011
A register	1011	1011	1011
Memory (MY)	0010	0010	0010

LD XP,r *Load r-register into XP*

Source Format: **LD XP,r**

Operation: XP ← r

OP-Code:

1	1	1	0	1	0	0	0	0	0	r_1	r_0

MSB LSB

E80H to E83H

Type: V

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the r-register into the 4-bit page part of index register IX.

Example:

	LD XP,B		LD XP,MX
XP register	1001	0001	1011
B register	0001	0001	0001
Memory (MX)	1011	1011	1011

LD Y,e

Load immediate data e into Y-register

Source Format: **LD Y,e**

Operation: YH ← e_7 to e_4, YL ← e_3 to e_0

OP-Code:

1	0	0	0	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0

MSB ... LSB — 800H to 8FFH

Type: I

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads 8-bit immediate data e into register Y.

Example:

	LD Y,E1H	
YH register	0001	1110
YL register	1100	0001

LD YH,r

Load r-register into YH

Source Format: **LD YH,r**

Operation: YH ← r

OP-Code:

1	1	1	0	1	0	0	1	0	1	r_1	r_0

MSB ... LSB — E94H to E97H

Type: V

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the r-register into the four high-order bits of register Y.

Example:

	LD YH,B		LD YH,MX
YH register	0000	0110	0101
B register	0110	0110	0110
Memory (MX)	0101	0101	0101

LD YL,r — *Load r-register into YL*

Source Format: **LD YL,r**

Operation: YL ← r

OP-Code:

1	1	1	0	1	0	0	1	1	0	r_1	r_0

MSB … LSB — E98H to E9BH

Type: V

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the r-register into the four low-order bits of register Y.

Example:

	LD YL,B		**LD YL,MX**
YL register	1011	1010	0111
B register	1010	1010	1010
Memory (MX)	0111	0111	0111

LD YP,r — *Load r-register into YP*

Source Format: **LD YP,r**

Operation: YP ← r

OP-Code:

1	1	1	0	1	0	0	1	0	0	r_1	r_0

MSB … LSB — E90H to E93H

Type: V

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the r-register into the 4-bit page part of index register IY.

Example:

	LD YP,MX		**LD YP,A**
YP register	0011	0000	0100
A register	0100	0100	0100
Memory (MX)	0000	0000	0000

NOP5 *No operation for 5 clock cycles*

Source Format: **NOP5**

Operation: No operation (5 clock cycles)

OP-Code:

1	1	1	1	1	1	1	1	1	0	1	1

MSB … LSB FFBH

Type: VI

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Increments the program counter by 1. Has no other effect for 5 clock cycles.

Example:

	NOP5	
PCB	0	0
PCP	0011	0011
PCS	0001 0011	0001 0100

NOP7 *No operation for 7 clock cycles*

Source Format: **NOP7**

Operation: No operation (7 clock cycles)

OP-Code:

1	1	1	1	1	1	1	1	1	1	1	1

MSB … LSB FFFH

Type: VI

Clock Cycles: 7

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Increments the program counter by 1. Has no other effect for 7 clock cycles.

Example:

	NOP7	
PCB	0	0
PCP	1010	1010
PCS	1001 1001	1001 1010

NOT r — *NOT r-register (one's complement)*

Source Format: **NOT r**

Operation: $r \leftarrow \overline{r}$

OP-Code:

MSB											LSB
1	1	0	1	0	0	r_1	r_0	1	1	1	1

D0FH to D3FH

Type: II

Clock Cycles: 7

Flag:
C – Not affected
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Performs a one's complement operation on the contents of the r-register.

Example:

	NOT A		NOT MY
A register	1001	0110	0110
Memory (MY)	1111	1111	0000
Z flag	0	0	1

OR r,i — *Logical OR immediate data i with r-register*

Source Format: **OR r,i**

Operation: $r \leftarrow r \vee i_3$ to i_0

OP-Code:

MSB											LSB
1	1	0	0	1	1	r_1	r_0	i_3	i_2	i_1	i_0

CC0H to CFFH

Type: II

Clock Cycles: 7

Flag:
C – Not affected
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Performs a logical OR operation between immediate data i and the contents of the r-register. The result is stored in the r-register.

Example:

	OR B,5		OR MX,0BH
B register	0100	0101	0101
Memory (MX)	0011	0011	0111
Z flag	0	0	0

OR r,q

Logical OR q-register with r-register

Source Format: **OR r,q**

Operation: r ← r ∨ q

OP-Code:

1	0	1	0	1	1	0	1	r_1	r_0	q_1	q_0

MSB LSB

AD0H to ADFH

Type: IV

Clock Cycles: 7

Flag:
C – Not affected
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Performs a logical OR operation between the contents of the q-register and the contents of the r-register. The result is stored in the r-register.

Example:

	OR MY,0		OR A,0CH
A register	0011	0011	1111
Memory (MY)	0000	0000	0000
Z flag	0	1	0

POP F

Pop stack data into flags

Source Format: **POP F**

Operation: F ← M(SP), SP ← SP + 1

OP-Code:

1	1	1	1	1	1	0	1	1	0	1	0

MSB LSB

FDAH

Type: VI

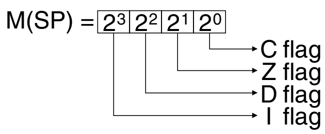
Clock Cycles: 5

Flag:
C – Set or Reset by M(SP) data
Z – Set or Reset by M(SP) data
D – Set or Reset by M(SP) data
I – Set or Reset by M(SP) data

Description: Replaces the flags (F) with the contents of the data memory location addressed by the stack pointer. SP is incremented by 1.

Example:

	POP F	
SP	C0	C1
Memory (C0H)	1001	1001
Flags (I,D,Z,C)	0001	1001



POP r — *Pop stack data into r-register*

Source Format: **POP r**

Operation: r ← M(SP), SP ← SP + 1

OP-Code:

1	1	1	1	1	1	0	1	0	0	r_1	r_0

MSB … LSB — FD0H to FD3H

Type: V

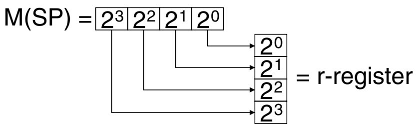
Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the data memory location addressed by the stack pointer into the r-register. SP is incremented by 1.

Example:

	POP B	
SP	C0	C1
Memory (C0H)	1001	1001
B register	0101	1001



POP XH — *Pop stack data into XH*

Source Format: **POP XH**

Operation: XH ← M(SP), SP ← SP + 1

OP-Code:

1	1	1	1	1	1	0	1	0	1	0	1

MSB … LSB — FD5H

Type: VI

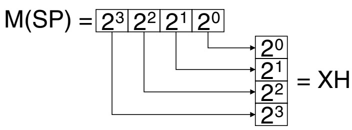
Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the data memory location addressed by the stack pointer into XH, the four high-order bits of X. SP is incremented by 1.

Example:

	POP XH	
SP	CE	CF
Memory (CEH)	0110	0110
XH register	0010	0110



POP XL — *Pop stack data into XL*

Source Format: **POP XL**

Operation: XL ← M(SP), SP ← SP + 1

OP-Code:

MSB											LSB
1	1	1	1	1	1	0	1	0	1	1	0

FD6H

Type: VI

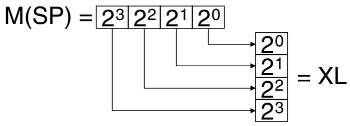
Clock Cycles: 5

Flag:
- **C** – Not affected
- **Z** – Not affected
- **D** – Not affected
- **I** – Not affected

Description: Loads the contents of the data memory location addressed by the stack pointer into XL, the four low-order bits of X. SP is incremented by 1.

Example:

	POP XL	
SP	C0	C1
Memory (C0H)	0001	0001
XL register	1101	0001



POP XP — *Pop stack data into XP*

Source Format: **POP XP**

Operation: XP ← M(SP), SP ← SP + 1

OP-Code:

MSB											LSB
1	1	1	1	1	1	0	1	0	1	0	0

FD4H

Type: VI

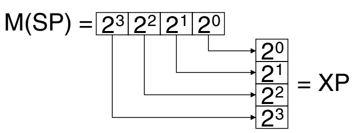
Clock Cycles: 5

Flag:
- **C** – Not affected
- **Z** – Not affected
- **D** – Not affected
- **I** – Not affected

Description: Loads the contents of the data memory location addressed by the stack pointer into XP, the 4-bit page part of IX. SP is incremented by 1.

Example:

	POP XP	
SP	B4	B5
Memory (B4H)	0101	0101
XP register	0111	0101



POP YH

Pop stack data into YH

Source Format: **POP YH**

Operation: YH ← M(SP), SP ← SP + 1

OP-Code:

1	1	1	1	1	1	0	1	1	0	0	0

MSB LSB

FD8H

Type: VI

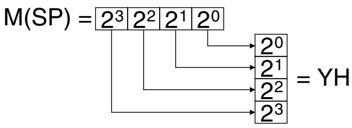
Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the data memory location addressed by the stack pointer into YH, the four high-order bits of Y. SP is incremented by 1.

Example:

	POP YH	
SP	C1	C2
Memory (C1H)	1101	1101
YH register	0010	1101



POP YL

Pop stack data into YL

Source Format: **POP YL**

Operation: YL ← M(SP), SP ← SP + 1

OP-Code:

1	1	1	1	1	1	0	1	1	0	0	1

MSB LSB

FD9H

Type: VI

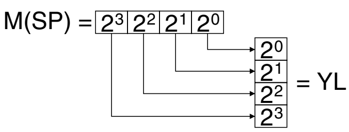
Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the data memory location addressed by the stack pointer into YL, the four low-order bits of Y. SP is incremented by 1.

Example:

	POP YL	
SP	CA	CB
Memory (CAH)	0100	0100
YL register	0101	0100



POP YP — *Pop stack data into YP*

Source Format: **POP YP**

Operation: YP ← M(SP), SP ← SP + 1

OP-Code:

MSB											LSB
1	1	1	1	1	1	0	1	0	1	1	1

FD7H

Type: VI

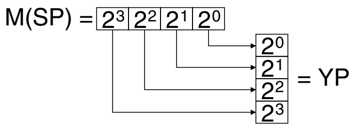
Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the contents of the data memory location addressed by the stack pointer into YP, the 4-bit page part of IY. SP is incremented by 1.

Example:

	POP YP	
SP	C0	C1
Memory (C0H)	0000	0000
YP register	0001	0000



PSET p — *Page set*

Source Format: **PSET p**

Operation: NBP ← p_4, NPP ← p_3 to p_0

OP-Code:

MSB											LSB
1	1	1	0	0	1	0	p_4	p_3	p_2	p_1	p_0

E40H to E5FH

Type: III

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads the most-significant bit of the 5-bit immediate data p to the new bank pointer (NBP) and the four low-order bits to the new page pointer (NPP).

Example:

	PSET 1FH		JP 00H
PCB	0	0	1
NBP	0	1	1
PCP	1000	1000	1111
NPP	0001	1111	1111
PCS	0010 0011	0010 0100	0000 0000

PUSH F *Push flag onto stack*

Source Format: **PUSH F**

Operation: SP' ← SP - 1, M(SP') ← F

OP-Code:

1	1	1	1	1	1	0	0	1	0	1	0

MSB LSB FCAH

Type: VI

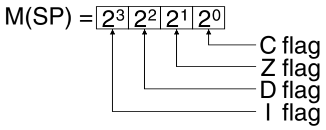
Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Decrements the stack pointer by 1 and loads the flags (F) into the data memory location addressed by SP.

Example:

	PUSH F	
SP	D0	CF
Memory (CFH)	0100	0001
Flags (I,D,Z,C)	0001	0001



PUSH r *Push r-register onto stack*

Source Format: **PUSH r**

Operation: SP' ← SP - 1, M(SP') ← r

OP-Code:

1	1	1	1	1	1	0	0	0	0	r_1	r_0

MSB LSB FC0H to FC3H

Type: V

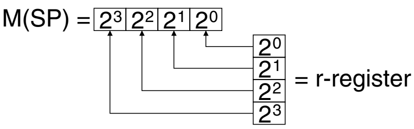
Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Decrements the stack pointer by 1 and loads the contents of the r-register into the data memory location addressed by SP.

Example:

	PUSH A	
SP	D0	CF
Memory (CFH)	1000	0010
A register	0010	0010



PUSH XH *Push XH onto stack*

Source Format: **PUSH XH**

Operation: SP' ← SP - 1, M(SP') ← XH

OP-Code:

1	1	1	1	1	1	0	0	0	1	0	1

MSB LSB

FC5H

Type: VI

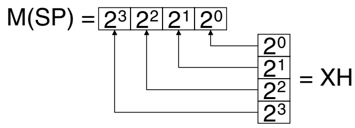
Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Decrements the stack pointer by 1 and loads the contents of XH, the four high-order bits of XHL, into the data memory location addressed by SP.

Example:

	PUSH XH	
SP	CC	CB
Memory (CBH)	0000	1000
XH register	1000	1000



PUSH XL *Push XL onto stack*

Source Format: **PUSH XL**

Operation: SP' ← SP - 1, M(SP') ← XL

OP-Code:

1	1	1	1	1	1	0	0	0	1	1	0

MSB LSB

FC6H

Type: VI

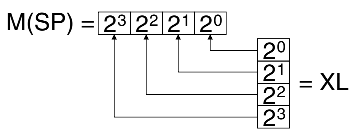
Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Decrements the stack pointer by 1 and loads the contents of XL, the four low-order bits of XHL, into the data memory location addressed by SP.

Example:

	PUSH XL	
SP	D0	CF
Memory (CFH)	1111	0110
XL register	0110	0110



PUSH XP *Push XP onto stack*

Source Format: **PUSH XP**

Operation: SP' ← SP - 1, M(SP') ← XP

OP-Code:

MSB											LSB
1	1	1	1	1	1	0	0	0	1	0	0

FC4H

Type: VI

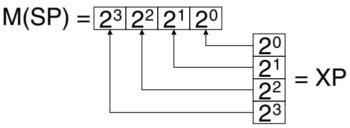
Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Decrements the stack pointer by 1 and loads the contents of XP, the page part of IX, into the data memory location addressed by SP.

Example:

	PUSH XP	
SP	D0	CF
Memory (CFH)	0011	0000
XP register	0000	0000



PUSH YH *Push YH onto stack*

Source Format: **PUSH YH**

Operation: SP' ← SP - 1, M(SP') ← YH

OP-Code:

MSB											LSB
1	1	1	1	1	1	0	0	1	0	0	0

FC8H

Type: VI

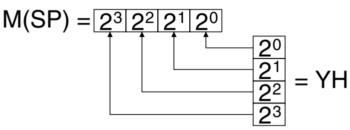
Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Decrements the stack pointer by 1 and loads the contents of YH, the four high-order bits of YHL, into the data memory location addressed by SP.

Example:

	PUSH YH	
SP	BF	BE
Memory (BEH)	0100	0001
YH register	0001	0001



PUSH YL *Push YL onto stack*

Source Format: **PUSH YL**

Operation: SP' ← SP - 1, M(SP') ← YL

OP-Code:

1	1	1	1	1	1	0	0	1	0	0	1

MSB … LSB — FC9H

Type: VI

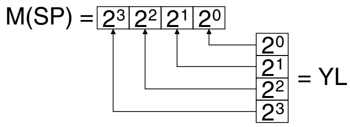
Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Decrements the stack pointer by 1 and loads the contents of YL, the four low-order bits of YHL, into the data memory location addressed by SP.

Example:

	PUSH YL	
SP	D0	CF
Memory (CFH)	0001	0111
YL register	0111	0111



PUSH YP *Push YP onto stack*

Source Format: **PUSH YP**

Operation: SP' ← SP - 1, M(SP') ← YP

OP-Code:

1	1	1	1	1	1	0	0	0	1	1	1

MSB … LSB — FC7H

Type: VI

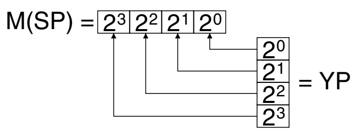
Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Decrements the stack pointer by 1 and loads the contents of YP, the page part of IY, into the data memory location addressed by SP.

Example:

	PUSH YP	
SP	C0	BF
Memory (BFH)	1111	0000
YP register	0000	0000



RCF Reset carry flag

Source Format: **RCF**

Operation: C ← 0

OP-Code:

1	1	1	1	0	1	0	1	1	1	1	0

MSB … LSB F5EH

Type: VI

Clock Cycles: 7

Flag:
C – Reset
Z – Not affected
D – Not affected
I – Not affected

Description: Resets the C (carry) flag.

Example:

	ADD A,4		RCF
A register	1101	0001	0001
C flag	0	1	0

RDF Reset decimal flag

Source Format: **RDF**

Operation: D ← 0

OP-Code:

1	1	1	1	0	1	0	1	1	0	1	1

MSB … LSB F5BH

Type: VI

Clock Cycles: 7

Flag:
C – Not affected
Z – Not affected
D – Reset
I – Not affected

Description: Resets the D (decimal) flag.

Example:

	ADD A,8	RDF	LD A,6	ADD A,8	
A register	0110	0100	0100	0110	1110
D flag	1	1	0	0	0
C flag	0	1	1	1	0
Z flag	0	0	0	0	0

RET — *Return from subroutine*

Source Format: **RET**

Operation: PCSL ← M(SP), PCSH ← M(SP+1), PCP ← M(SP+2), SP ← SP + 3

OP-Code:

1	1	1	1	1	1	0	1	1	1	1	1

MSB … LSB   FDFH

Type: VI

Clock Cycles: 7

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Jumps to the return address that was pushed onto the stack when the subroutine was called.

Example:

	RET	
PCP	1101	0010
PCS	1000 1101	0010 1101
SP	BD	C0
Memory (SP)	1101	1101
Memory (SP+1)	0010	0010
Memory (SP+2)	0010	0010

RETD e — *Load immediate data e to memory, and increment X by 2, then return*

Source Format: **RETD e**

Operation: PCSL ← M(SP), PCSH ← M(SP+1), PCP ← M(SP+2), SP ← SP + 3,
M(X) ← e_3 to e_0, M(X+1) ← e_7 to e_4, X ← X + 2

OP-Code:

0	0	0	1	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0

MSB … LSB   100H to 1FFH

Type: I

Clock Cycles: 12

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Loads 8-bit immediate data e into the data memory location addressed by IX and executes the RET command. X is incremented by 2.

Example:

	RETD F5H	
PCP	0000	0010
PCS	1010 1011	0010 1101
SP	BD	C0
Memory (SP)	1101	1101
Memory (SP+1)	0010	0010
Memory (SP+2)	0010	0010
X register	0010 1010	0010 1100
Memory (2AH)	0000	0101
Memory (2BH)	0000	1111

RETS — *Return then skip an instruction*

Source Format: **RETS**

Operation: PCSL ← M(SP), PCSH ← M(SP+1), PCP ← M(SP+2), SP ← SP + 3, PC ← PC + 1

OP-Code:

1	1	1	1	1	1	0	1	1	1	1	0
MSB											LSB

FDEH

Type: VI

Clock Cycles: 12

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Jumps to the return address that was pushed onto the stack when the subroutine was called and then skips one instruction.

Example:

	RETS	
PCP	0110	0000
PCS	1001 0000	0000 0111
SP	B0	B3
Memory (SP)	0110	0110
Memory (SP+1)	0000	0000
Memory (SP+2)	0000	0000

RLC r — *Rotate r-register left with carry*

Source Format: **RLC r**

Operation: $d_3 \leftarrow d_2$, $d_2 \leftarrow d_1$, $d_1 \leftarrow d_0$, $d_0 \leftarrow C$, $C \leftarrow d_3$

OP-Code:

1	0	1	0	1	1	1	1	r_1	r_0	r_1	r_0
MSB											LSB

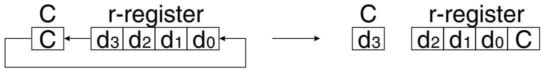
AF0H to AFFH

Type: IV

Clock Cycles: 7

Flag:
C – Set when the high-order bit of the r-register is 1; otherwise, reset.
Z – Not affected
D – Not affected
I – Not affected

Description: Shifts the contents of the r-register one bit to the left. The high-order bit is shifted into the carry flag and the carry bit becomes the low-order bit of the r-register.



Example:

	RLC A	
A register	0011	0111
C flag	1	0

RRC r — *Rotate r-register right with carry*

Source Format: **RRC r**

Operation: $d_3 \leftarrow C$, $d_2 \leftarrow d_3$, $d_1 \leftarrow d_2$, $d_0 \leftarrow d_1$, $C \leftarrow d_0$

OP-Code:

MSB											LSB
1	1	1	0	1	0	0	0	1	1	r_1	r_0

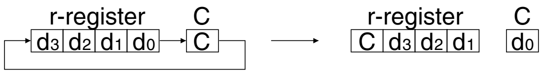
E8CH to E8FH

Type: V

Clock Cycles: 5

Flag:
C – Set when the low-order bit of the r-register is 1; otherwise, reset.
Z – Not affected
D – Not affected
I – Not affected

Description: Shifts the contents of the r-register one bit to the right. The low-order bit is shifted into the carry flag and the carry bit becomes the high-order bit of the r-register.



Example:

	RRC MY	
Memory (MY)	1010	1101
C flag	1	0

RST F,i — *Reset flags using immediate data i*

Source Format: **RST F,i**

Operation: $F \leftarrow F \wedge i_3$ to i_0

OP-Code:

MSB											LSB
1	1	1	1	0	1	0	1	i_3	i_2	i_1	i_0

F50H to F5FH

Type: IV

Clock Cycles: 7

Flag:
C – Reset if i_0 is zero; otherwise, not affected.
Z – Reset if i_1 is zero; otherwise, not affected.
D – Reset if i_2 is zero; otherwise, not affected.
I – Reset if i_3 is zero; otherwise, not affected.

Description: Performs a logical AND operation between immediate data i and the contents of the flags. The result is stored in each respective flag.

Example:

	RST F,2	
Flags (I,D,Z,C)	1010	0010

RZF — *Reset zero flag*

Source Format: **RZF**

Operation: Z ← 0

OP-Code:

1	1	1	1	0	1	0	1	1	1	0	1

MSB … LSB — F5DH

Type: VI

Clock Cycles: 7

Flag:
- **C** – Not affected
- **Z** – Reset
- **D** – Not affected
- **I** – Not affected

Description: Resets the Z (zero) flag.

Example:

	ADD A,3		RZF
Z flag	0	1	0
A register	1101	0000	0000

SBC r,i — *Subtract with carry immediate data i from r-register*

Source Format: **SBC r,i**

Operation: r ← r - i_3 to i_0 - C

OP-Code:

1	1	0	1	0	1	r_1	r_0	i_3	i_2	i_1	i_0

MSB … LSB — D40H to D7FH

Type: II

Clock Cycles: 7

Flag:
- **C** – Set if a borrow is generated; otherwise, reset.
- **Z** – Set if the result is zero; otherwise, reset.
- **D** – Not affected
- **I** – Not affected

Description: Subtracts the carry flag and immediate data i from the r-register.

Example:

	SBC A,9		SBC MY,0DH
A register	1000	1111	1111
Memory (MY)	1110	1110	0000
C flag	0	1	0
Z flag	0	0	1

SBC r,q

Subtract with carry q-register from r-register

Source Format: **SBC r,q**

Operation: r ← r - q - C

OP-Code:

1	0	1	0	1	0	1	1	r_1	r_0	q_1	q_0

MSB LSB

AB0H to ABFH

Type: IV

Clock Cycles: 7

Flag:
C – Set if a borrow is generated; otherwise, reset.
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Subtracts the carry flag and the contents of the q-register from the r-register.

Example:

	SBC A,B		SBC MY,MX
A register	1110	1011	1011
B register	0010	0010	0010
Memory (MX)	1001	1001	1001
Memory (MY)	0100	0100	1011
C flag	1	0	1
Z flag	0	0	0

SCF

Set carry flag

Source Format: **SCF**

Operation: C ← 1

OP-Code:

1	1	1	1	0	1	0	0	0	0	0	1

MSB LSB

F41H

Type: VI

Clock Cycles: 7

Flag:
C – Set
Z – Not affected
D – Not affected
I – Not affected

Description: Sets the C (carry) flag.

Example:

	SCF	
C flag	0	1

SCPX MX,r *Subtract with carry r-register from M(X) and increment X by 1*

Source Format: **SCPX MX,r**

Operation: M(X) ← M(X) - r - C, X ← X + 1

OP-Code:

1	1	1	1	0	0	1	1	1	0	r_1	r_0

MSB … LSB — F38H to F3BH

Type: V

Clock Cycles: 7

Flag:
C – Set if a borrow is generated; otherwise, reset.
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Subtracts the carry flag and the contents of the r-register from the data memory location addressed by IX. X is incremented by 1. Incrementing X does not affect the flags.

Example:

	SCPX MX,B	
X register	0101 0000	0101 0001
Memory (50H)	0110	0100
B register	0010	0010
C flag	0	0
Z flag	0	0

SCPY MY,r *Subtract with carry r-register from M(Y) and increment Y by 1*

Source Format: **SCPY MY,r**

Operation: M(Y) ← M(Y) - r - C, Y ← Y + 1

OP-Code:

1	1	1	1	0	0	1	1	1	1	r_1	r_0

MSB … LSB — F3CH to F3FH

Type: V

Clock Cycles: 7

Flag:
C – Set if a borrow is generated; otherwise, reset.
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Subtracts the carry flag and the contents of the r-register from the data memory location addressed by IY. Y is incremented by 1. Incrementing Y does not affect the flags.

Example:

	SCPY MY,A	
Y register	1111 1111	0000 0000
Memory (FFH)	0111	0100
A register	0010	0010
C flag	1	0
Z flag	1	0

SDF *Set decimal flag*

Source Format: **SDF**

Operation: D ← 1

OP-Code:

1	1	1	1	0	1	0	0	0	1	0	0

MSB LSB F44H

Type: VI

Clock Cycles: 7

Flag:
C – Not affected
Z – Not affected
D – Set
I – Not affected

Description: Sets the D (decimal) flag.

Example:

	SDF	
D flag	0	1

SET F,i *Set flags using immediate data i*

Source Format: **SET F,i**

Operation: F ← F ∨ i_3 to i_0

OP-Code:

1	1	1	1	0	1	0	0	i_3	i_2	i_1	i_0

MSB LSB F40H to F4FH

Type: IV

Clock Cycles: 7

Flag:
C – Set if i_0 is 1; otherwise, not affected.
Z – Set if i_1 is 1; otherwise, not affected.
D – Set if i_2 is 1; otherwise, not affected.
I – Set if i_3 is 1; otherwise, not affected.

Description: Performs a logical OR operation between immediate data i and the contents of the flags. The results are stored in each respective flag.

Example:

	SET F,0DH	
Flags (C,Z,D,I)	0011	1111

SLP *Sleep*

Source Format: **SLP**

Operation: Stop CPU and peripheral oscillator

OP-Code:

1	1	1	1	1	1	1	1	1	0	0	1

MSB LSB FF9H

Type: VI

Clock Cycles: 5

Flag:
C – Not affected
Z – Not affected
D – Not affected
I – Not affected

Description: Stops the CPU and the peripheral oscillator. When an interrupt occurs PCP and PCS are pushed onto the stack as the return address and the interrupt service routine is executed.

Example:

	Instruction	State	PCP	PCS	I flag
		RUN	0100	0011 0000	1
	SLP		0100	0011 0001	1
		SLEEP			
Interrupt →					
	NOP5	RUN			
			0001	0000 0001	0

SUB r,q *Subtract q-register from r-register*

Source Format: **SUB r,q**

Operation: r ← r - q

OP-Code:

1	0	1	0	1	0	1	0	r_1	r_0	q_1	q_0

MSB LSB AA0H to AAFH

Type: IV

Clock Cycles: 7

Flag:
C – Set if a borrow is generated; otherwise, reset.
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Subtracts the contents of the q-register from the r-register.

Example:

	SUB A,B	
A register	1100	1001
B register	0011	0011
C flag	1	0
Z flag	0	0

SZF *Set zero flag*

Source Format: **SZF**

Operation: Z ← 1

OP-Code:

1	1	1	1	0	1	0	0	0	0	1	0

MSB ... LSB — F42H

Type: VI

Clock Cycles: 7

Flag:
C – Not affected
Z – Set
D – Not affected
I – Not affected

Description: Sets the Z (zero) flag.

Example:

	SZF	
Z flag	0	1

XOR r,i *Exclusive-OR immediate data i with r-register*

Source Format: **XOR r,i**

Operation: r ← r ∀ i_3 to i_0

OP-Code:

1	1	0	1	0	0	r_1	r_0	i_3	i_2	i_1	i_0

MSB ... LSB — D00H to D3FH

Type: II

Clock Cycles: 7

Flag:
C – Not affected
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Performs an exclusive-OR operation between immediate data i and the contents of the r-register. The result is stored in the r-register.

Example:

	XOR A,12		XOR MX,1
A register	0110	1010	1010
Memory (MX)	0001	0001	0000
Z flag	0	0	1

XOR r,q — *Exclusive-OR q-register with r-register*

Source Format: **XOR r,q**

Operation: $r \leftarrow r \forall q$

OP-Code:

1	0	1	0	1	1	1	0	r_1	r_0	q_1	q_0

MSB ... LSB

AE0H to AEFH

Type: IV

Clock Cycles: 7

Flag:

C – Not affected
Z – Set if the result is zero; otherwise, reset.
D – Not affected
I – Not affected

Description: Performs an exclusive-OR operation between the contents of the q-register and the contents of the r-register. The result is stored in the r-register.

Example:

	XOR A,MY		**XOR MX,B**
A register	0100	1100	1100
B register	1111	1111	1111
Memory (MX)	0111	0111	1000
Memory (MY)	1000	1000	1000
Z flag	0	0	0

ABBREVIATIONS

A .... A register (4 bits)
B .... B register (4 bits)
M(SP) ..... Contents of the data memory location whose address is specified by stack pointer SP (4 bits)
M(X) .... Contents of the data memory location whose address is specified by IX (4 bits)
M(Y) .... Contents of the data memory location whose address is specified by IY (4 bits)
M($n_{3\text{-}0}$) .... Contents of the data memory location within the register area 00H to 0FH (4 bits)
MX .... Data memory location whose address is specified by IX
MY .... Data memory location whose address is specified by IY
NBP .... New Bank Pointer (1 bit)
NPP .... New Page Pointer (4 bits)
PCB .... Program Counter-Bank (1 bit)
PCP .... Program Counter-Page (4 bits)
PCS .... Program Counter-Step (8 bits)
PCSH ..... Four high-order bits of PCS
PCSL ...... Four low-order bits of PCS
RP .... Register Pointer (4 bits)
SP .... Stack Pointer (8 bits)
SPH .... Four high-order bits of SP
SPL .... Four low-order bits of SP
X .... Eight low-order bits of IX, that is, XHL
XH .... Four high-order bits of X
XL .... Four low-order bits of X
XP .... Four high-order bits of IX (page part)
Y .... Eight low-order bits of IY, that is, YHL
YH .... Four high-order bits of Y
YL .... Four low-order bits of Y
YP .... Four high-order bits of IY (page part)
+ .... Addition
– .... Subtraction
$\wedge$ .... Logical AND
$\vee$ .... Logical OR
$\forall$ .... Exclusive-OR
$\downarrow$ .... Reset flag
$\uparrow$ .... Set flag
$\updownarrow$ .... Set/reset flag
$*$ .... Decimal addition/subtraction

APPENDIX A. E0C6200A (ADVANCED E0C6200) CORE CPU

E0C6200A is an improved version of the E0C6200. In this section, E0C6200A is described only in terms of its differences with E0C6200. It is recommended that users of E0C6200A read this section.

E0C6200A is a Core CPU which has been made easier to integrate software by improving the parts of the E0C6200 CPU which are difficult to use.
This section lists its differences with E0C6200; for items which are not included here, refer to the corresponding section in this manual.

A1 Outline of Differences

- The D (decimal) flag is set to "0" during initial reset.
- Modifications of the interrupt circuit
  - The interrupt timing has been shifted to 0.5 clock later.
  - ***<Reference>*** In the 1-chip micro controller which uses E0C6200A, writing on the interrupt mask register and reading the interrupt factor flag during EI (enable interrupt flag) are possible. (However, consult the respective hardware manuals to find out whether these are possible with the CPU peripheral circuits.)

A2 Detailed Description of the Differences

A2.1 Initial reset

The D (decimal) flag will be set as follows through initial reset:

Table A2.1.1 D (decimal) flag initial setting

CPU Core	E0C6200A	E0C6200
D (decimal) flag setting	0	Undefined

Owing to this, bugs due to omission of D (decimal) flag setting during software development can now be easily prevented.

For the values of other registers and flags during initial reset, see Section 2.5.4, "Initial reset".

A2.2 Interrupt

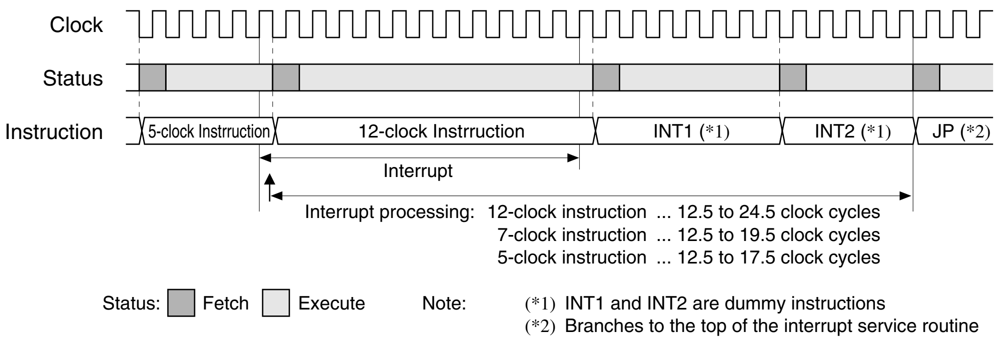
Operation during interrupt issuance

The time it takes to complete interrupt processing by hardware after the Core CPU receives the interrupt request has changed as follows:

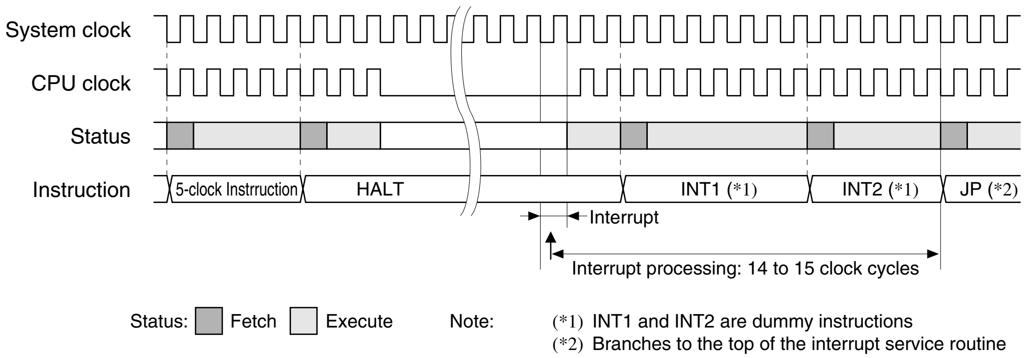
Table A2.2.1 Required interrupt processing time

Item		E0C6200A (clock cycles)	E0C6200 (clock cycles)
a) During instruction execution	12-cycle instruction execution	12.5 to 24.5	13 to 25
	7-cycle instruction execution	12.5 to 19.5	13 to 20
	5-cycle instruction execution	12.5 to 17.5	13 to 18
b) At HALT mode		14 to 15	14 to 15
c) During PSET instruction execution	PSET + CALL	12.5 to 24.5	13 to 25
	PSET + JP	12.5 to 22.5	13 to 23

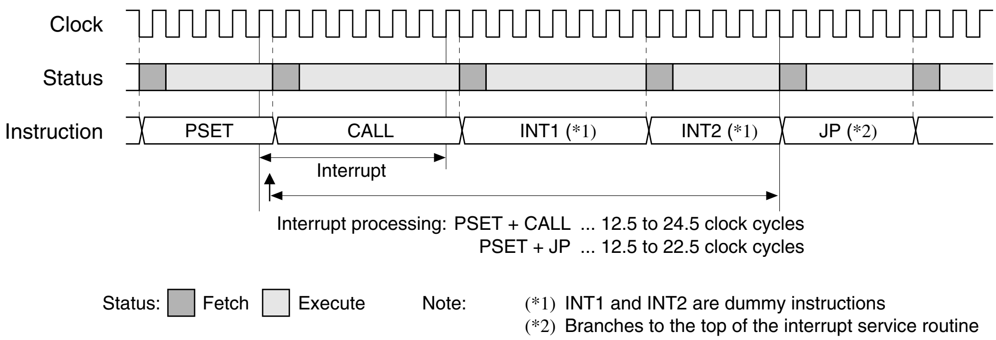
Figure A2.2.1 shows the timing chart of the E0C6200A interrupt.



a) During instruction execution



b) At HALT mode



c) During "PSET" instruction execution

Fig. A2.2.1 Timing chart of E0C6200A interrupt

<Reference 1> Writing on the interrupt mask register during EI

This section describes the operation for writing on the interrupt mask register during EI (enable interrupt flag) in the regular 1-chip micro controller which uses E0C6200 Core CPU and in the regular 1-chip micro controller which uses E0C6200A Core CPU. For information on accurate operation, see the respective hardware manuals of the E0C62 Family.

Table A2.2.2 Writing on the interrupt mask register at EI

CPU Core	E0C6200A	E0C6200
Writing on the interrupt mask register at EI	Possible	Not possible

The operation during the instruction execution for writing "0" (i.e., to mask the interrupt factor) on the interrupt mask register at EI is shown in Figure A2.2.2. At this point, the interrupt is masked 0.5 clock before the start of the instruction execution through the 0.5 clock advance operation.
Moreover, during the instruction execution for writing "1" (i.e., to cancel the interrupt mask) on the mask register at EI, it is the same as the ordinary interrupt timing as shown in Figure A2.2.2. In other words, if the interrupt factor flag value is set to "1", the interrupt processing by hardware will start in the next instruction execution cycle 0.5 clock before the completion of the instruction execution.

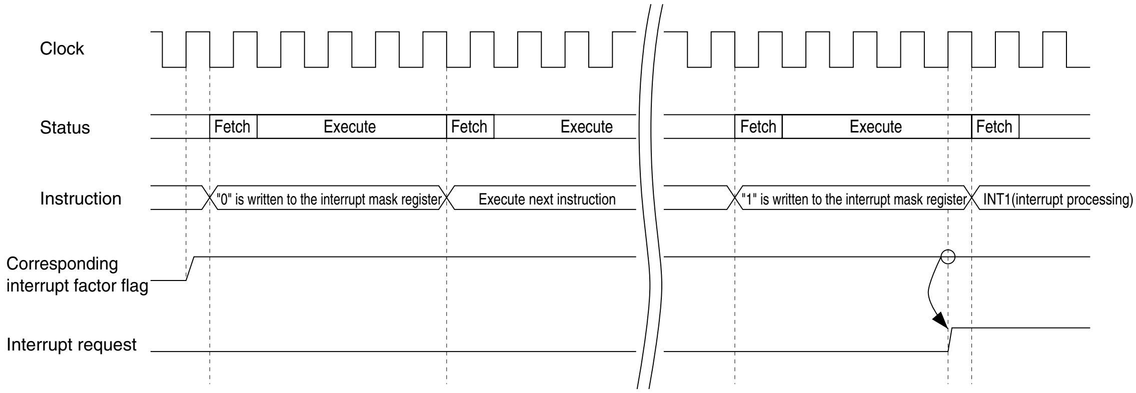


Fig. A2.2.2 Writing on the interrupt mask register and interrupt request generation

<Reference 2> Reading the interrupt factor flag during EI

This section describes the operation for reading the interrupt factor flag during EI (enable interrupt flag) in the regular 1-chip micro controller which uses E0C6200 Core CPU and in the regular 1-chip micro controller which uses E0C6200A Core CPU. For information on accurate operation, see the respective hardware manuals of the E0C62 Family.

Table A2.2.3 Reading the interrupt factor flag at EI

CPU Core	E0C6200A	E0C6200
Reading the interrupt factor flag at EI	Possible	Not possible

At EI, reading the interrupt factor flag is possible but caution must be observed in the following case: when the value of the interrupt mask register corresponding to the interrupt factor flag which is to be read is set to "1" (unmasked). In this case, interrupt request may be issued to the CPU due to the timing by which the interrupt factor flag is set to "1", or the interrupt factor flag may be cleared by reading it and hence interrupt request will not be issued.
Particularly when there are multiple interrupt factor flags in the same address, extra caution is required.

APPENDIX B. INSTRUCTION INDEX

EPSON International Sales Operations

AMERICA

EPSON ELECTRONICS AMERICA, INC.

- HEADQUARTERS -
1960 E. Grand Avenue
El Segundo, CA 90245, U.S.A.
Phone: +1-310-955-5300 Fax: +1-310-955-5400

- SALES OFFICES -

West
150 River Oaks Parkway
San Jose, CA 95134, U.S.A.
Phone: +1-408-922-0200 Fax: +1-408-922-0238

Central
101 Virginia Street, Suite 290
Crystal Lake, IL 60014, U.S.A.
Phone: +1-815-455-7630 Fax: +1-815-455-7633

Northeast
301 Edgewater Place, Suite 120
Wakefield, MA 01880, U.S.A.
Phone: +1-781-246-3600 Fax: +1-781-246-5443

Southeast
3010 Royal Blvd. South, Suite 170
Alpharetta, GA 30005, U.S.A.
Phone: +1-877-EEA-0020 Fax: +1-770-777-2637

EUROPE

EPSON EUROPE ELECTRONICS GmbH

- HEADQUARTERS -
Riesstrasse 15
80992 Munich, GERMANY
Phone: +49-(0)89-14005-0 Fax: +49-(0)89-14005-110

- GERMANY -
SALES OFFICE
Altstadtstrasse 176
51379 Leverkusen, GERMANY
Phone: +49-(0)2171-5045-0 Fax: +49-(0)2171-5045-10

- UNITED KINGDOM -
UK BRANCH OFFICE
Unit 2.4, Doncastle House, Doncastle Road
Bracknell, Berkshire RG12 8PE, ENGLAND
Phone: +44-(0)1344-381700 Fax: +44-(0)1344-381701

- FRANCE -
FRENCH BRANCH OFFICE
1 Avenue de l' Atlantique, LP 915 Les Conquerants
Z.A. de Courtaboeuf 2, F-91976 Les Ulis Cedex, FRANCE
Phone: +33-(0)1-64862350 Fax: +33-(0)1-64862355

ASIA

- CHINA -
EPSON (CHINA) CO., LTD.
28F, Beijing Silver Tower 2# North RD DongSanHuan
ChaoYang District, Beijing, CHINA
Phone: 64106655 Fax: 64107319

SHANGHAI BRANCH
4F, Bldg., 27, No. 69, Gui Jing Road
Caohejing, Shanghai, CHINA
Phone: 21-6485-5552 Fax: 21-6485-0775

- HONG KONG, CHINA -
EPSON HONG KONG LTD.
20/F., Harbour Centre, 25 Harbour Road
Wanchai, HONG KONG
Phone: +852-2585-4600 Fax: +852-2827-4346
Telex: 65542 EPSCO HX

- TAIWAN -
EPSON TAIWAN TECHNOLOGY & TRADING LTD.
10F, No. 287, Nanking East Road, Sec. 3
Taipei, TAIWAN
Phone: 02-2717-7360 Fax: 02-2712-9164
Telex: 24444 EPSONTB

HSINCHU OFFICE
13F-3, No. 295, Kuang-Fu Road, Sec. 2
HsinChu 300, TAIWAN
Phone: 03-573-9900 Fax: 03-573-9169

- SINGAPORE -
EPSON SINGAPORE PTE., LTD.
No. 1 Temasek Avenue, #36-00
Millenia Tower, SINGAPORE 039192
Phone: +65-337-7911 Fax: +65-334-2716

- KOREA -
SEIKO EPSON CORPORATION KOREA OFFICE
50F, KLI 63 Bldg., 60 Yoido-dong
Youngdeungpo-Ku, Seoul, 150-763, KOREA
Phone: 02-784-6027 Fax: 02-767-3677

- JAPAN -
SEIKO EPSON CORPORATION
ELECTRONIC DEVICES MARKETING DIVISION

Electronic Device Marketing Department
IC Marketing & Engineering Group
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5816 Fax: +81-(0)42-587-5624

ED International Marketing Department Europe & U.S.A.
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5812 Fax: +81-(0)42-587-5564

ED International Marketing Department Asia
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5814 Fax: +81-(0)42-587-5110

In pursuit of **"Saving" Technology**, Epson electronic devices.
Our lineup of semiconductors, liquid crystal displays and quartz devices
assists in creating the products of our customers' dreams.
Epson IS energy savings.

EPSON

SEIKO EPSON CORPORATION
ELECTRONIC DEVICES MARKETING DIVISION

■ EPSON Electronic Devices Website
http://www.epson.co.jp/device/

First issue FEBRUARY 1989, Printed JANUARY 1998 in Japan Ⓜ A