SECTION ZI

Microprocessor Architecture	ZI-1
Z-80 CPU Architecture	ZI-2
Z-80 CPU Pin-Out	ZI-5
Comparing The Z-80, 8080 and 6800 Microprocessors	ZI-7
Z-80 CPU Timing	ZI-10

$\overline{\text{RFSH}}$ Output, active low. $\overline{\text{RFSH}}$ indicates that the lower 7 bits of the address bus contain a refresh address for dynamic memories. The current $\overline{\text{MREQ}}$ signal should be used to do a refresh read of all dynamic memories.

$\overline{\text{HALT}}$ Output, active low. $\overline{\text{HALT}}$ indicates that the CPU has executed a HALT software instruction and is waiting for a non-maskable or maskable interrupt to resume operation.

$\overline{\text{WAIT}}$ Input, active low. $\overline{\text{WAIT}}$ indicates to the CPU that the addressed memory or I/O devices are not ready for data transfer.

$\overline{\text{INT}}$ Input, active low. The Interrupt Request signal is generated by I/O devices. A request will be honored at the end of the current instruction if the internal interrupt enable FF (IFF) is enabled.

$\overline{\text{NMI}}$ Input, negative edge triggered. The non-maskable interrupt line has a higher priority than $\overline{\text{INT}}$ and is always recognized at the end of the current instruction

$\overline{\text{RESET}}$ Input, active low. $\overline{\text{RESET}}$ forces the program counter to zero and initializes the CPU.

$\overline{\text{BUSRQ}}$ Input, active low. The bus request signal is used to request the CPU address bus, data bus and tri-state output signals to go to a high impedance state so that other devices can control these buses.

$\overline{\text{BUSAK}}$ Output, active low. Bus acknowledge is used to indicate to the requesting device that the buses and control signals have been set to their high impedance state.

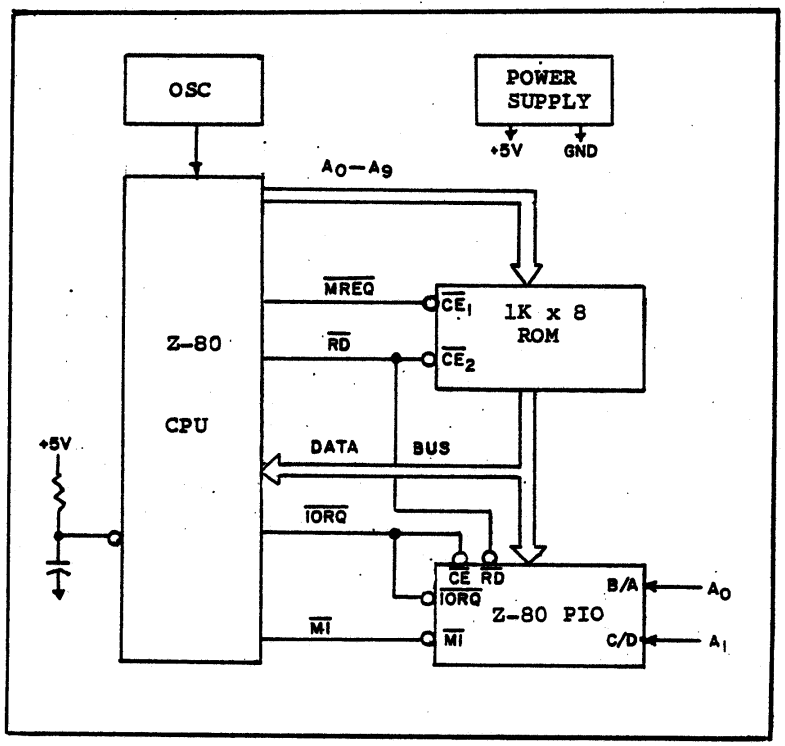
Φ Single phase TTL clock.

COMPARING THE Z-80, 8080 AND 6800 MICROPROC$SSORS

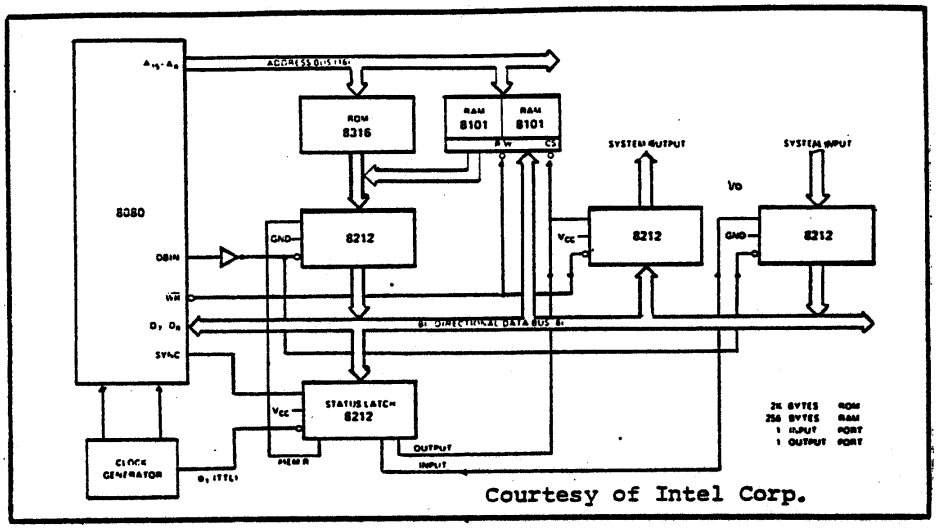
The 8080 and the 6800 are two of the most popular microprocessors on the market today. Incorporated in the Z-80 architecture and instruction set are some of the best features of both of these popular machines. We will make a brief comparison of these processors at this point.

Minimum Systems

One method of comparing microprocessors is by the amount of hardware required for a minimum system.

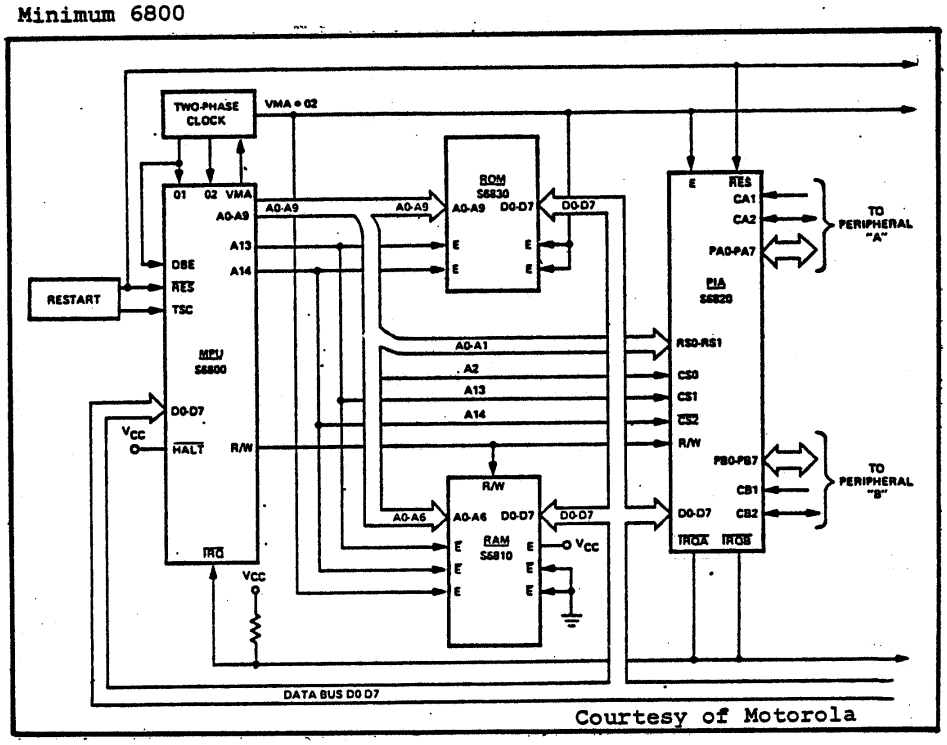


Minimum 8080



Courtesy of Intel Corp.

Minimum 6800



Courtesy of Motorola

Comparison of Z-80, 8080 and 6800 CPU Chips

A. Similarities

1. 8 and 16 bit addressing capabilities
2. Stack addressing
3. 65K of directly addressable memory space
4. Interrupt capabilities
5. Tristate address and data bus buffering
6. A 40 pin CPU chip
7. N-channel MOS technology

B. Differences	Z-80	8080	6800
No. of Instructions	158	78	72
8 bit registers	14	7	2
16 bit registers	8	5	2
Index registers	2	0	1
Address Modes	10	7	8
I/O addresses	256	256	*
Flag Bits	6	5	6
Voltage requirements	+5V	+5V, -5V, +12V	+5V
Non Maskable Interrupt	Yes	No	Yes
TTL compatible inputs	Yes	No	No
Asynchronous inputs	Yes	No	No
3 state control lines	Yes	No	Yes
Clock phases	One	Two	Two
Static operation	Yes	No	No
Dynamic memory refresh	Yes	No	No

C. Instruction Capabilities

8080

The 8080 has a simple instruction set. Lack of relative and index addressing capabilities, however, limits its flexibility to some degree. The 8080 is most efficiently programmed by making extensive use of stack save operations in conjunction with subroutine operations. A variety of conditional Call and Return instructions are available for this purpose.

6800

The 6800 instruction set is characterized by extensive use of read/write memory, both for programmable register operations and for I/O. The 6800 uses memory mapped I/O exclusively. The 6800 has one of the largest varieties of Branch on condition instructions. For relatively simple programs the 6800

is extremely effective because of the large variety of memory reference instructions. For the more complex programs, however, the lack of directly addressable registers, such as those available in the 8080, limit its usefulness to a degree. It should be noted that the 6800 limits relative direct addressing to unconditional branch instructions.

Z-80

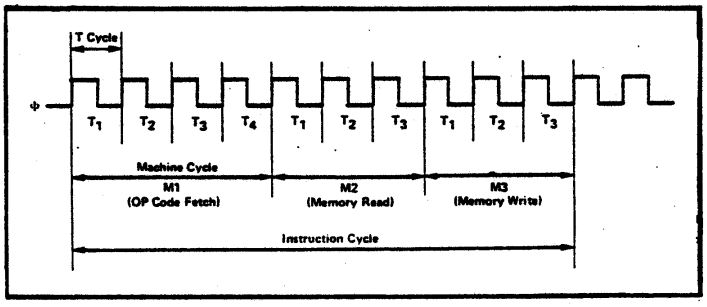
The Z-80 has all of the capabilities of 8080. In addition, it has both program relative addressing, extremely useful for conditional branch operations, and index addressing. Index addressing as used in the Z-80, has two advantages, it simplifies block move and search routines, and in memory reference instruction makes the working register, which would often have to be used as memory pointers available for data operations.

Z-80 CPU TIMING

All instructions consist of a sequence of the following basic operations:

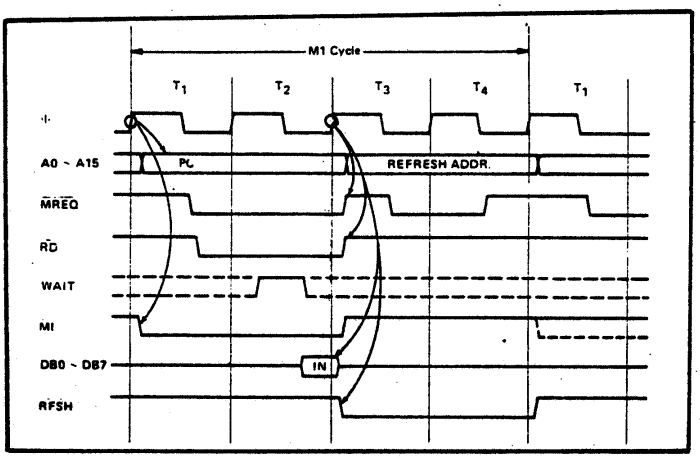
Memory Read or Write
I/O Device Read or Write
Interrupt Acknowledge

Each basic operation takes from three to six clock periods, although any operation can be lengthened with wait states. Basic operations are referred to as an M (machine) cycle clock periods are called T states. A basic CPU Timing example is illustrated.

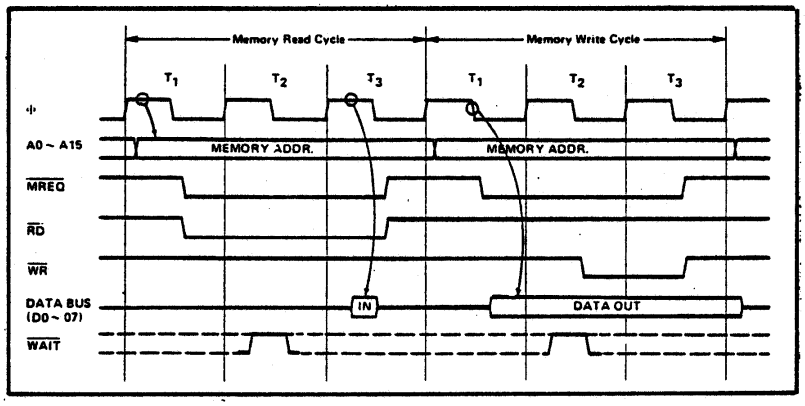


BASIC CPU TIMING

The diagrams which follow illustrate the timing of basic CPU operations.

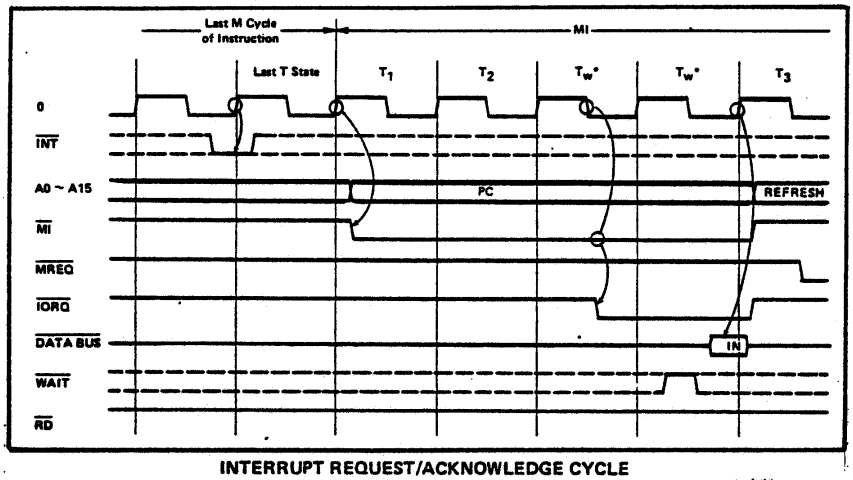


INSTRUCTION OP CODE FETCH

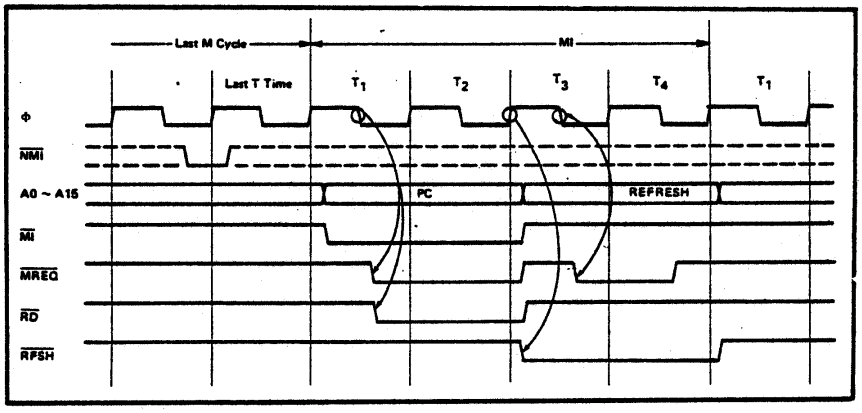


MEMORY READ OR WRITE CYCLES

Courtesy Zilog Corp.

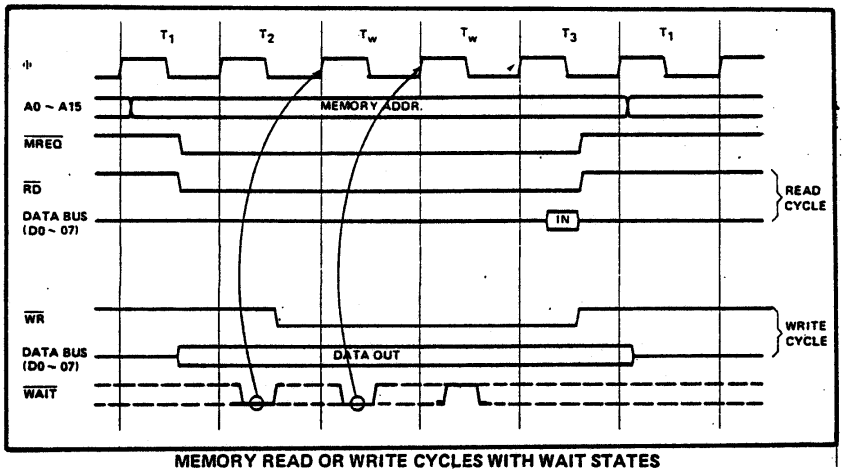


INTERRUPT REQUEST/ACKNOWLEDGE CYCLE

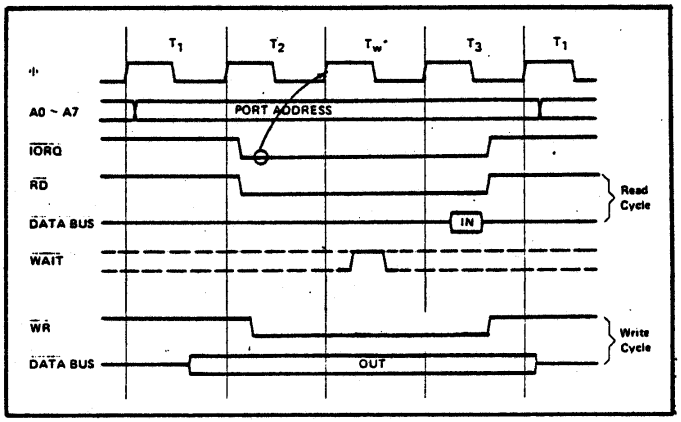


NON MASKABLE INTERRUPT REQUEST OPERATION

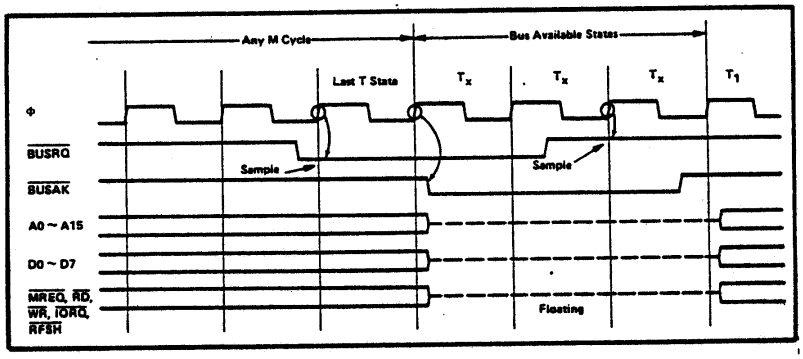
Courtesy Zilog Corp.



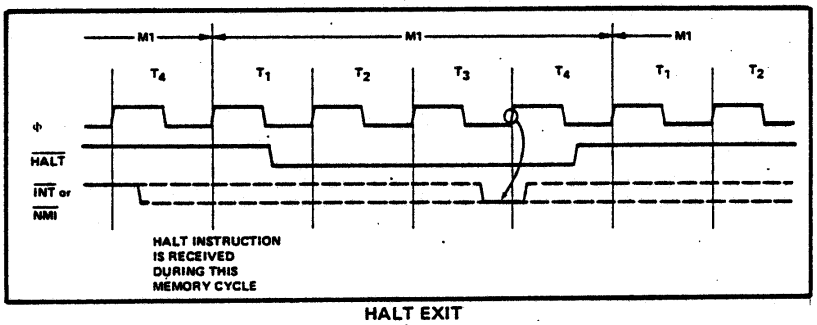
MEMORY READ OR WRITE CYCLES WITH WAIT STATES



INPUT OR OUTPUT CYCLES



BUS REQUEST/ACKNOWLEDGE CYCLE



HALT EXIT

Courtesy Zilog Corp.

SECTION ZPI

Introduction To Programming ZPI-1

Z-80 Addressing Modes ZPI-2

The Z-80 Instruction Set ZPI-5

E

INTRODUCTION TO PROGRAMMING

Levels of Programming

As we will illustrate , the instruction set for a particular microprocessor is a list of mnemonic statements. Each mnemonic identifies one of the instructions in the set. Associated with each instruction mnemonic are one or more bytes of data (binary words) which are the actual "machine language" instructions. These binary coded instructions are decoded by the CPU which initiates the necessary commands and timing signals to execute the instructions. Machine language is the only form of instruction to which the machine can respond. No matter what level of language is used by the programmer, it must ultimately be converted to machine language.

Machine Language Programming

A programmer may enter his program into the machine in the form of ones and zeros of the machine code. Generally this is a slow tedious process.

Hand Assembly

A program can be written in the instruction mnemonics. Each instruction would then be coded in the octal or hexadecimal equivalent of the machine code and be entered into the machine via a key board. The coding procedure is known as hand assembly.

Assembly Language Programming

A more efficient form of programming is Assembly language programming. In this form of programming an assembler, either resident in the machine or available as a cross-assembler, takes over the task of machine language coding and assignment of the memory locations. In the field of dedicate microcontrollers assembly language programming is the most efficient method to use.

Programming in Higher Order Languages

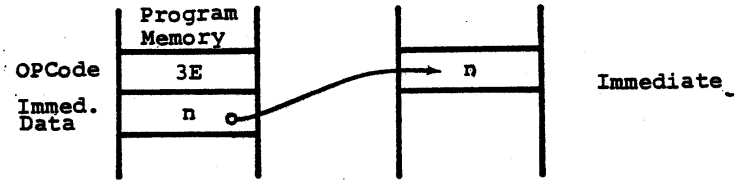
Higher languages such as Basic, PLM, Algol, Fortran IV, Cobol and APL are conversational languages, as such, they are simple to use. The programmer must have a thorough knowledge of the syntax of the language, but he need not have any knowledge of the architecture of the machine to be used.

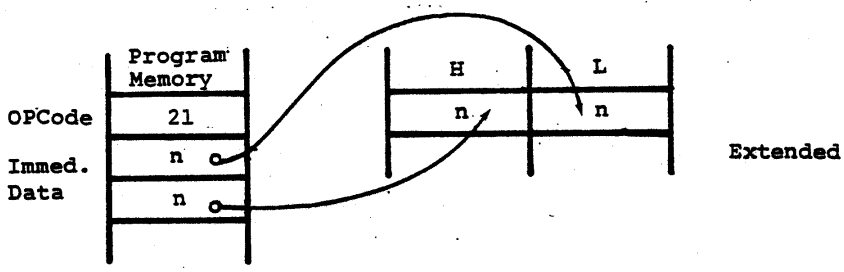
Addressing Modes

Z-80 instructions operate on data in register, in external memory, space and auxiliary memory space (I/O space). Depending on the type of instruction, different modes of addressing are used. This section summarizes the addressing modes used in the Z-80.

Immediate and Immediate Extended

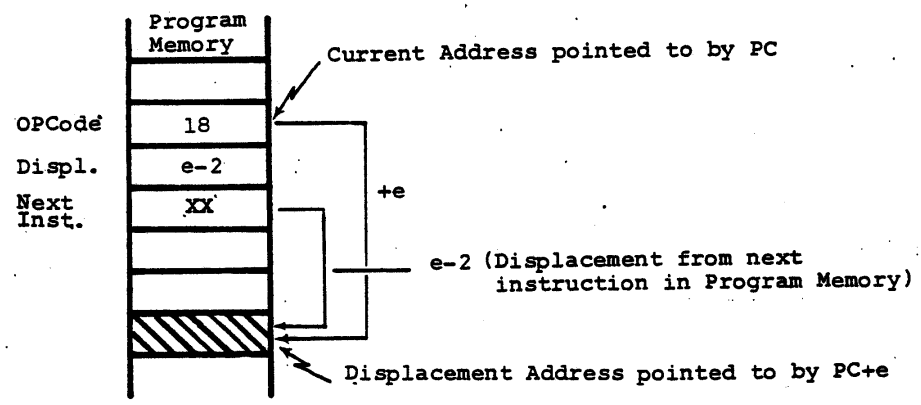
In these modes of addressing the byte of data, or two bytes of data immediately following the opcode in program memory are operated upon by the instruction.





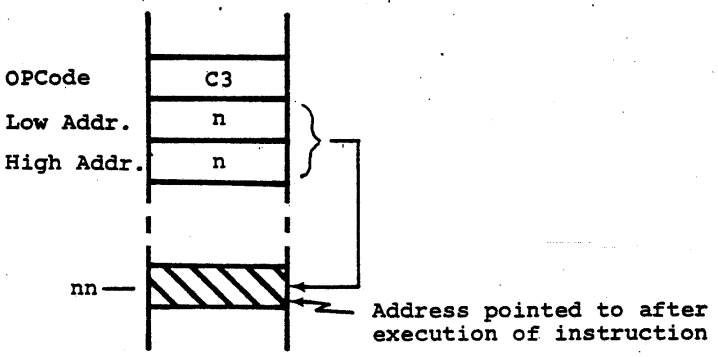
Relative Addressing

Relative addressing is a mode used in jump or branch instructions. One byte of data following the opcode specifies a displacement (e) which is added to the current contents of the program counter. The displacement has a range between +127 and -128.



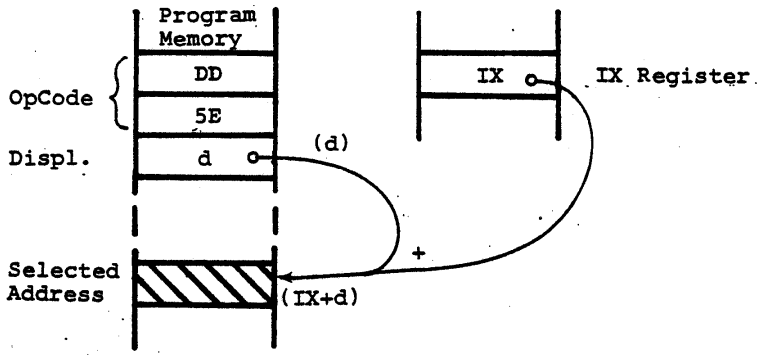
Extended Addressing

Extended addressing is required for program jumps to any location in memory space, or to load or store data in any location in memory space. The actual source or destination is specified in the instruction as a two byte address (nn).



Indexed Addressing

In this type of instruction the byte of data following the second byte of the OpCode is added to the current contents of a specified index register (IX or IY) to form a memory pointer which points to any desired location in memory space. (Contents of the index register is not altered by execution of this instruction.



Register Addressing

Many Z-80 OpCodes contain selected bits that specify which CPU register to be used in the execution of an instruction such as a register to register data move. These are one byte instructions.

Implied Addressing

Implied addressing involves instructions where the OpCode automatically implies one or more CPU registers as containing the operands. The accumulator is the implied destination in arithmetic operations

Register Indirect

This mode of addressing uses a register pair (such as the HL) as a pointer to any location in memory. In the register indirect mode the register pair used as a pointer is indicated by parenthesis. For example the symbol (HL) indicates that the contents of the HL register pair are to be used as a pointer to a memory location.

Bit Addressing

Bit Addressing implies that the Opcode of a Bit Set, Reset or Test operation will operate on a single bit (specified by the OpCode) at an address location. The method of addressing may be register, register indirect, or indexed.

Z-80 MICROPROCESSOR
FUNDAMENTALS AND APPLICATIONS

THE Z-80 INSTRUCTION SET - (MAPPED)

8-BIT LOAD OPERATIONS

Source

Dest.

8 BIT 'LD'		IMPLIED		REGISTER							REG INDIRECT			INDEXED		EXT. ADDR.	IMME.
		I	R	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX + d)	(IY + d)	(nn)	n
REGISTER	A	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n
	B			47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n
	C			4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n
	D			57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n
	E			5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n
	H			67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n
	L			6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n
REG INDIRECT	(HL)			77	70	71	72	73	74	75							36 n
	(BC)			02													
	(DE)			12													
INDEXED	(IX+d)			DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n
	(IY+d)			FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n
EXT. ADDR	(nn)			32 n n													
IMPLIED	I			ED 47													
	R			ED 4F													

Courtesy Zilog Corp.

16-BIT LOAD OPERATIONS

Source

Dest,

16 BIT 'LD'		REGISTER							IMM. EXT.	EXT. ADDR.	REG. INDIR.
		AF	BC	DE	HL	SP	IX	IY	nn	(nn)	(SP)
REGISTER	AF										F1
	BC								01 n n	ED 4B n n	C1
	DE								11 n n	ED 5B n n	D1
	HL								21 n n	2A n n	E1
	SP				F9		DD F9	FD F9	31 n n	ED 7B n n	
	IX								DD 21 n n	DD 2A n n	DD E1
	IY								FD 21 n n	FD 2A n n	FD E1
EXT. ADDR.	(nn)		ED 43 n n	ED 53 n n	22 n n	ED 73 n n	DD 22 n n	FD 22 n n			↑ POP INST
REG. IND.	(SP)	F5	C5	D5	E5		DD E5	FD E5	← PUSH INSTR		

8-BIT ARITHMETIC AND LOGIC OPERATIONS

8 BIT ARITH AND LOGIC	REGISTER ADDRESSING							REG. INDIR.	INDEXED		IMMED.
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
'ADD'	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
ADD w CARRY 'ADC'	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
SUBTRACT 'SUB'	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
SUB w CARRY 'SBC'	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
'AND'	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
'XOR'	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
'OR'	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
COMPARE 'CP'	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
INCREMENT 'INC'	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DECREMENT 'DEC'	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

16-BIT ARITHMETIC OPERATIONS

16 BIT ARITHMETIC		BC	DE	HL	SP	IX	IY
'ADD'	HL	09	19	29	39		
	IX	DD 09	DD 19		DD 39	DD 29	
	IY	FD 09	FD 19		FD 39		FD 29
ADD WITH CARRY AND SET FLAGS 'ADC'	HL	ED 4A	ED 5A	ED 6A	ED 7A		
SUB WITH CARRY AND SET FLAGS 'SBC'	HL	ED 42	ED 52	ED 62	ED 72		
INCREMENT 'INC'		03	13	23	33	DD 23	FD 23
DECREMENT 'DEC'		0B	1B	2B	3B	DD 2B	FD 2B

GENERAL PURPOSE AF	
Decimal Adjust Acc, 'DAA'	27
Complement Acc, 'CPL'	2F
Negate Acc, 'NEG' (2's complement)	ED 44
Complement Carry Flag, 'CCF'	3F
Set Carry Flag, 'SCF'	37

Block Transfers and Searches

BLOCK TRANSFER		REG. INDIR. (HL)	Reg HL points to source Reg DE points to destination Reg BC is byte counter
REG. INDIR.	(DE)	ED A0	'LDI' – Load (DE)⟵(HL) Inc HL & DE, Dec BC
		ED B0	'LDIR' – Load (DE)⟵(HL) Inc HL & DE, Dec BC, Repeat until BC = 0
		ED A8	'LDD' – Load (DE)⟵(HL) Dec HL & DE, Dec BC
		ED B8	'LDDR' – Load (DE)⟵(HL) Dec HL & DE, Dec BC, Repeat until BC = 0

REG. INDIR.	BLOCK SEARCH
(HL)	HL points to location in memory to be compared with accumulator BC is byte counter
ED A1	'CPI' Inc HL, Dec BC
ED B1	'CPIR', Inc HL, Dec BC repeat until BC = 0 or find match
ED A9	'CPD' Dec HL & BC
ED B9	'CPDR' Dec HL & BC Repeat until BC = 0 or find match

Jumps, Calls and Returns

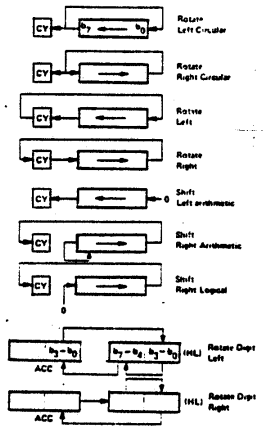
JUMP, CALL and RETURN			UN-COND.	CARRY	NON CARRY	ZERO	NON ZERO	PARITY EVEN	PARITY ODD	SIGN NEG	SIGN POS	REG B=0
JUMP 'JP'	IMMED. EXT.	nn	C3 n n	DA n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
JUMP 'JR'	RELATIVE	PC+e	18 e-2	38 e-2	30 e-2	28 e-2	20 e-2					
JUMP 'JP'	REG. INDIR.	(HL)	E9									
JUMP 'JP'		(IX)	DD E9									
JUMP 'JP'		(IY)	FD E9									
'CALL'	IMMED. EXT.	nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
DECREMENT B, JUMP IF NON ZERO 'DJNZ'	RELATIVE	PC+e										10 e-2
RETURN 'RET'	REGISTER INDIR.	(SP) (SP+1)	C9	D8	D0	C8	C0	E8	E0	F8	F0	
RETURN FROM INT 'RETI'	REG. INDIR.	(SP) (SP+1)	ED 4D									
RETURN FROM NON MASKABLE INT 'RETN'	REG. INDIR.	(SP) (SP+1)	ED 45									

Courtesy Zilog Corp.

ROTATE AND SHIFT OPERATIONS

ROTATES AND SHIFTS										
	A	B	C	D	E	H	L	(HL)	(IX + d)	(IY + d)
'RLC'	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
'RRC'	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
'RL'	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
'RR'	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
'SLA'	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 26	FD CB d 26
'SRA'	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
'SRL'	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
'RLD'								ED 6F		
'RRD'								ED 67		

	A
RLCA	07
RRCA	0F
RLA	17
RRA	1F



EXCHANGE AND RESTART OPERATIONS

'EX' 'EXX'		IMPLIED ADDRESSING				
		AF	BC, DE & HL	HL	IX	IY
IMPLIED	AF	08				
	BC, DE & HL		D9			
	DE			EB		
REG. INDIR.	(SP)			E3	DD E3	FD E3

RESTART		OP CODE	
CALL ADDRESS	0000_H	C7	'RST 0'
	0008_H	CF	'RST 8'
	0010_H	D7	'RST 16'
	0018_H	DF	'RST 24'
	0020_H	E7	'RST 32'
	0028_H	EF	'RST 40'
	0030_H	F7	'RST 48'
	0038_H	FF	'RST 56'

INPUT/OUTPUT OPERATIONS AND CPU CONTROL

INPUT	PORT ADDRESS		IMMED. (n)	REG. INDIR. (C)
INPUT 'IN'	REG ADDRESSING	A	DB	ED 78
		B		ED 40
		C		ED 48
		D		ED 50
		E		ED 58
		H		ED 60
		L		ED 68
'INI' – INPUT & Inc HL, Dec B	REG. INDIR	(HL)		ED A2
'INIR' – INP, Inc HL, Dec B, REPEAT IF B≠0				ED B2
'IND' – INPUT & Dec HL, Dec B				ED AA
'INDR' – INPUT, Dec HL, Dec B, REPEAT IF B≠0				ED BA

CPU CONTROL	
'NOP'	00
'HALT'	76
DISABLE INT '(DI)'	F3
ENABLE INT '(EI)'	FB
SET INT MODE 0 'IM0'	ED 46
SET INT MODE 1 'IM1' 0038_H	ED 56
SET INT MODE 2 'IM2'	ED 5E

OUTPUT	PORT DESTINATION ADDRESS		REGISTER A	B	C	D	E	H	L	REG. IND. (HL)
'OUT'	IMMED.	(n)	D3							
	REG. IND.	(C)	ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69	
'OUTI' – OUTPUT Inc HL, Dec b	REG. IND.	(C)								ED A3
'OTIR' – OUTPUT, Inc HL, Dec B, REPEAT IF B≠0	REG. IND.	(C)								ED B3
'OUTD' – OUTPUT Dec HL & B	REG. IND.	(C)								ED AB
'OTDR' – OUTPUT, Dec HL & B, REPEAT IF B≠0	REG. IND.	(C)								ED BB

Courtesy Zilog Corp.

BIT SET, RESET AND TEST

BIT OPS	BIT	REGISTER ADDRESSING							REG. INDIR.	INDEXED	
		A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
TEST 'BIT'	0	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
	1	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
	2	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
	3	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
	4	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
	5	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
	6	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
	7	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
RESET BIT 'RES'	0	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
	1	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
	2	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
	3	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
	4	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
	5	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
	6	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
	7	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET BIT 'SET'	0	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
	1	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
	2	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
	3	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
	4	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
	5	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
	6	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
	7	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

Courtesy Zilog Corp.

SECTION ZIN

Programming The Z-80	ZIN-1
Load Instructions	ZIN-1
Block Transfers and Searches	ZIN-9
Arithmetics	ZIN-14
Rotates and Shifts	ZIN-18
Bit Set, Reset and Test	ZIN-19
Jump Instructions	ZIN-20
Calls and Returns	ZIN-24
Input/Output Instructions	ZIN-27
Decimal to Hexadecimal Conversions	ZIN-31

PROGRAMMING THE Z-80

To effectively program the Z-80, we must first learn what each instruction does. The best approach will be to examine the instructions in small groups, using them in exercises as we proceed.

8 Bit Load Instructions

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	OP-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T Cycles	Comments
LD r, r'	r ← r'	•	•	•	•	•	•	01 r r'	1	1	4	r, r' Reg.
LD r, n	r ← n	•	•	•	•	•	•	00 r 110 ← n →	2	2	7	000 B 001 C
LD r, (HL)	r ← (HL)	•	•	•	•	•	•	01 r 110	1	2	7	010 D
LD r, (IX+d)	r ← (IX+d)	•	•	•	•	•	•	11 011 101 01 r 110 ← d →	3	5	19	011 E 100 H 101 L
LD r, (IY+d)	r ← (IY+d)	•	•	•	•	•	•	11 111 101 01 r 110 ← d →	3	5	19	111 A
LD (HL), r	(HL) ← r	•	•	•	•	•	•	01 110 r	1	2	7	
LD (IX+d), r	(IX+d) ← r	•	•	•	•	•	•	11 011 101 01 110 r ← d →	3	5	19	
LD (IY+d), r	(IY+d) ← r	•	•	•	•	•	•	11 111 101 01 110 r ← d →	3	5	19	
LD (HL), n	(HL) ← n	•	•	•	•	•	•	00 110 110 ← n →	2	3	10	
LD (IX+d), n	(IX+d) ← n	•	•	•	•	•	•	11 011 101 00 110 110 ← d → ← n →	4	5	19	
LD (IY+d), n	(IY+d) ← n	•	•	•	•	•	•	11 111 101 00 110 110 ← d → ← n →	4	5	19	
LD A, (BC)	A ← (BC)	•	•	•	•	•	•	00 001 010	1	2	7	
LD A, (DE)	A ← (DE)	•	•	•	•	•	•	00 011 010	1	2	7	
LD A, (nn)	A ← (nn)	•	•	•	•	•	•	00 111 010 ← n → ← n →	3	4	13	
LD (BC), A	(BC) ← A	•	•	•	•	•	•	00 000 010	1	2	7	
LD (DE), A	(DE) ← A	•	•	•	•	•	•	00 010 010	1	2	7	
LD (nn), A	(nn) ← A	•	•	•	•	•	•	00 110 010 ← n → ← n →	3	4	13	
LD A, I	A ← I	•	‡	IFF	‡	0	0	11 101 101 01 010 111	2	2	9	
LD A, R	A ← R	•	‡	IFF	‡	0	0	11 101 101 01 011 111	2	2	9	
LD I, A	I ← A	•	•	•	•	•	•	11 101 101 01 000 111	2	2	9	
LD R, A	R ← A	•	•	•	•	•	•	11 101 101 01 001 111	2	2	9	

Notes:

r, r' means any of the registers A, B, C, D, E, H, L

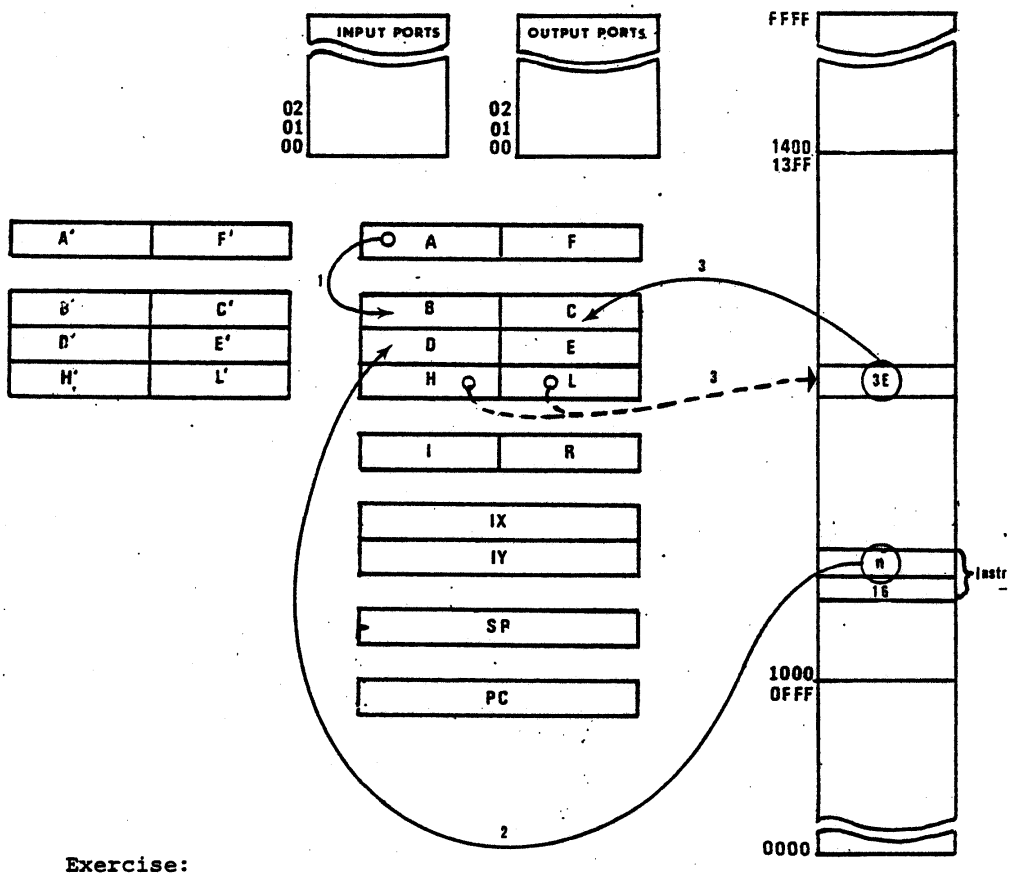
IFF the content of the interrupt enable flip-flop (IFF) is copied into the P/V flag

Flag Notation:

• = flag not affected,
0 = flag reset, 1 = flag set,
X = flag is unknown,
‡ = flag is affected according to the result of the operation.

Register Load Instructions

1. LD B,A — 1 Byte, Register to Register Load.
2. LD D,n — 2 Byte, Load Immediate.
3. LD C,(HL) — 3 Byte, Register Load, Indirect.



Exercise:

1. Write mnemonic instructions to transfer data in the A register into the H register.
2. Transfer data from memory location 1ØØAH to 1ØØBH.
3. Initialize Registers D with $(200)_{10}$ and E with 01001100B.

Many of the 16-bit load instructions of the Z-80 are identical to those available on the 8080, however, additional instructions are available for moving data to and from sixteen bit registers such as the IX, IY, and the Stack Pointer.

16 Bit Load Instructions

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments	
LD dd, nn	dd ← nn	•	•	•	•	•	•	00 dd0 001	3	3	10	dd	Pair
								← n →				00	BC
								← n →				01	DE
LD IX, nn	IX ← nn	•	•	•	•	•	•	11 011 101	4	4	14	10	HL
								00 100 001				11	SP
								← n →					
								← n →					
LD IY, nn	IY ← nn	•	•	•	•	•	•	11 111 101	4	4	14		
								00 100 001					
								← n →					
								← n →					
LD HL, (nn)	H ← (nn+1)	•	•	•	•	•	•	00 101 010	3	5	16		
	L ← (nn)							← n →					
								← n →					
LD dd, (nn)	dd_H ← (nn+1)	•	•	•	•	•	•	11 101 101	4	6	20		
	dd_L ← (nn)							01 dd1 011					
								← n →					
								← n →					
LD IX, (nn)	IX_H ← (nn+1)	•	•	•	•	•	•	11 011 101	4	6	20		
	IX_L ← (nn)							00 101 010					
								← n →					
								← n →					
LD IY, (nn)	IY_H ← (nn+1)	•	•	•	•	•	•	11 111 101	4	6	20		
	IY_L ← (nn)							00 101 010					
								← n →					
								← n →					
LD (nn), HL	(nn+1) ← H	•	•	•	•	•	•	00 100 010	3	5	16		
	(nn) ← L							← n →					
								← n →					
LD (nn), dd	(nn+1) ← dd_H	•	•	•	•	•	•	11 101 101	4	6	20		
	(nn) ← dd_L							01 dd0 011					
								← n →					
								← n →					
LD (nn), IX	(nn+1) ← IX_H	•	•	•	•	•	•	11 011 101	4	6	20		
	(nn) ← IX_L							00 100 010					
								← n →					
								← n →					
LD (nn), IY	(nn+1) ← IY_H	•	•	•	•	•	•	11 111 101	4	6	20		
	(nn) ← IY_L							00 100 010					
								← n →					
								← n →					
LD SP, HL	SP ← HL	•	•	•	•	•	•	11 111 001	1	1	6		
LD SP, IX	SP ← IX	•	•	•	•	•	•	11 011 101	2	2	10		
								11 111 001					
LD SP, IY	SP ← IY	•	•	•	•	•	•	11 111 101	2	2	10		
								11 111 001				qq	Pair
PUSH qq	(SP-2) ← qq_L	•	•	•	•	•	•	11 qq0 101	1	3	11	00	BC
	(SP-1) ← qq_H											01	DE
PUSH IX	(SP-2) ← IX_L	•	•	•	•	•	•	11 011 101	2	4	15	10	HL
	(SP-1) ← IX_H							11 100 101				11	AF
PUSH IY	(SP-2) ← IY_L	•	•	•	•	•	•	11 111 101	2	4	15		
	(SP-1) ← IY_H							11 100 101					
POP qq	qq_H ← (SP+1)	•	•	•	•	•	•	11 qq0 001	1	3	10		
	qq_L ← (SP)												
POP IX	IX_H ← (SP+1)	•	•	•	•	•	•	11 011 101	2	4	14		
	IX_L ← (SP)							11 100 001					
POP IY	IY_H ← (SP+1)	•	•	•	•	•	•	11 111 101	2	4	14		
	IY_L ← (SP)							11 100 001					

Notes:

dd is any of the register pairs BC, DE, HL, SP
qq is any of the register pairs AF, BC, DE, HL

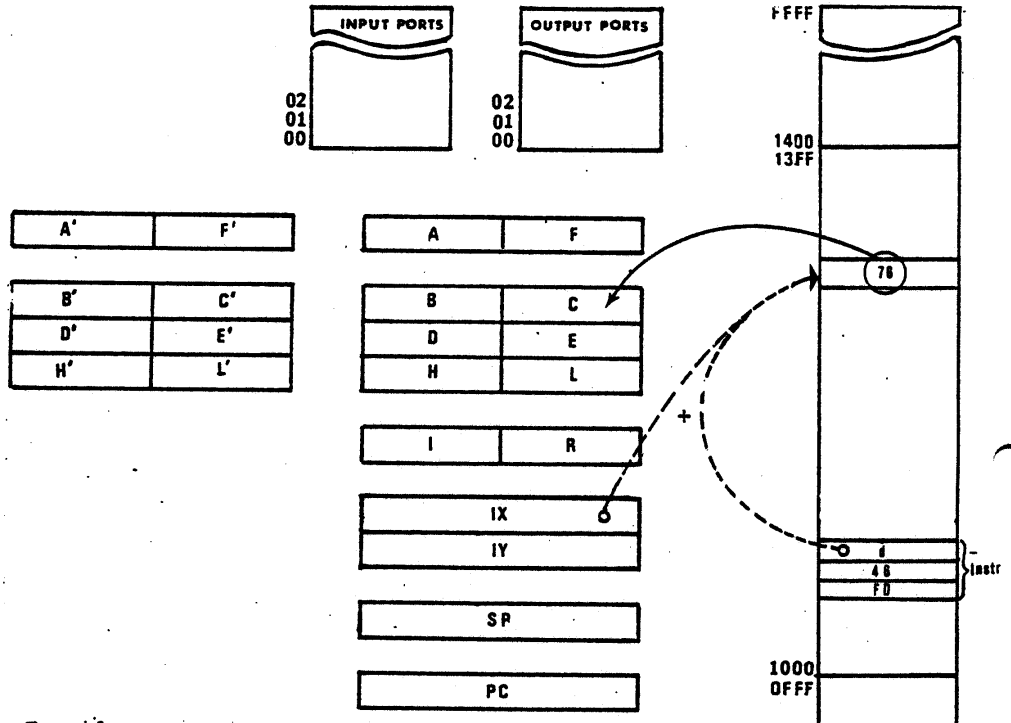
$(PAIR)_H$, $(PAIR)_L$ refer to high order and low order eight bits of the register pair respectively.
E.g. BC_L = C, AF_H = A

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown, ↕ flag is affected according to the result of the operation.

Courtesy Zilog Corporation

Register Load, Indexed

LD C,(IX+d) 3 Byte, Indexed Load



Example

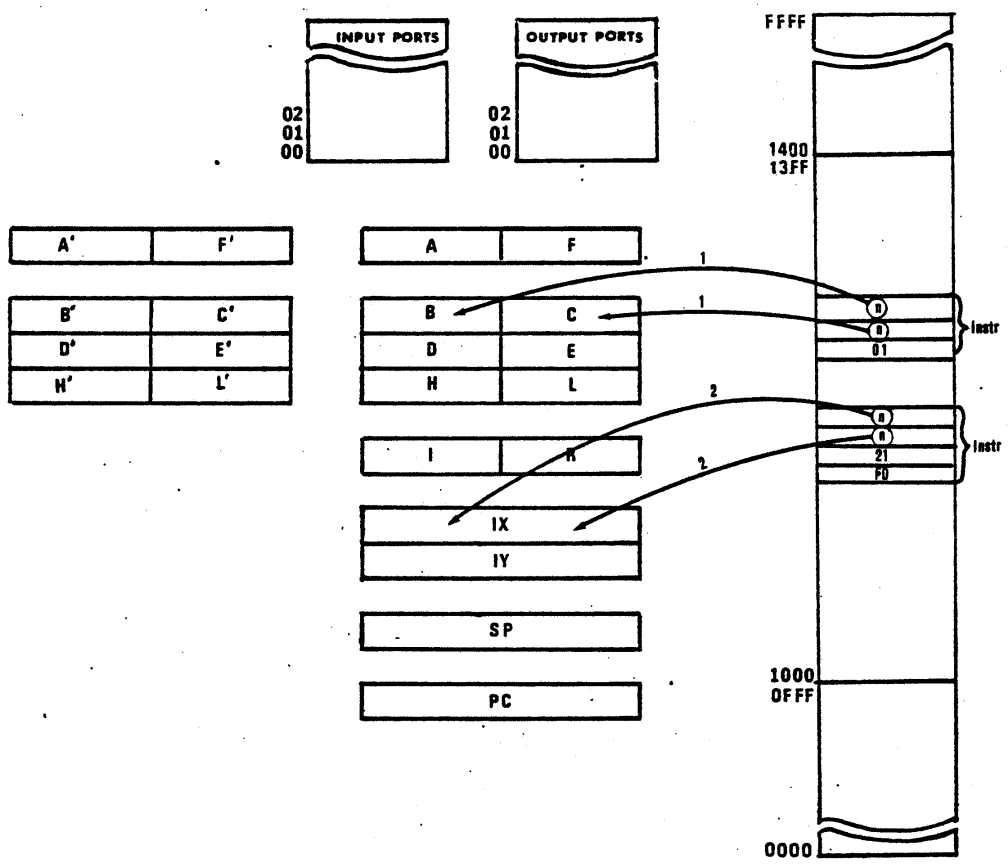
Assume the contents of the IX register to be 1300
Load the contents of memory location 1334 into the
C register.

LD C,(IX+d)	DD
--	4E
--	34

Exercise: If contents of IY is 1Ø4Ø, write mnemonics to transfer data from 100A to 100B.

16 Bit Load Instructions

1. LD dd,nn 3 Byte, Load Immediate
2. LD IX,nn 4 Byte, Load Immediate



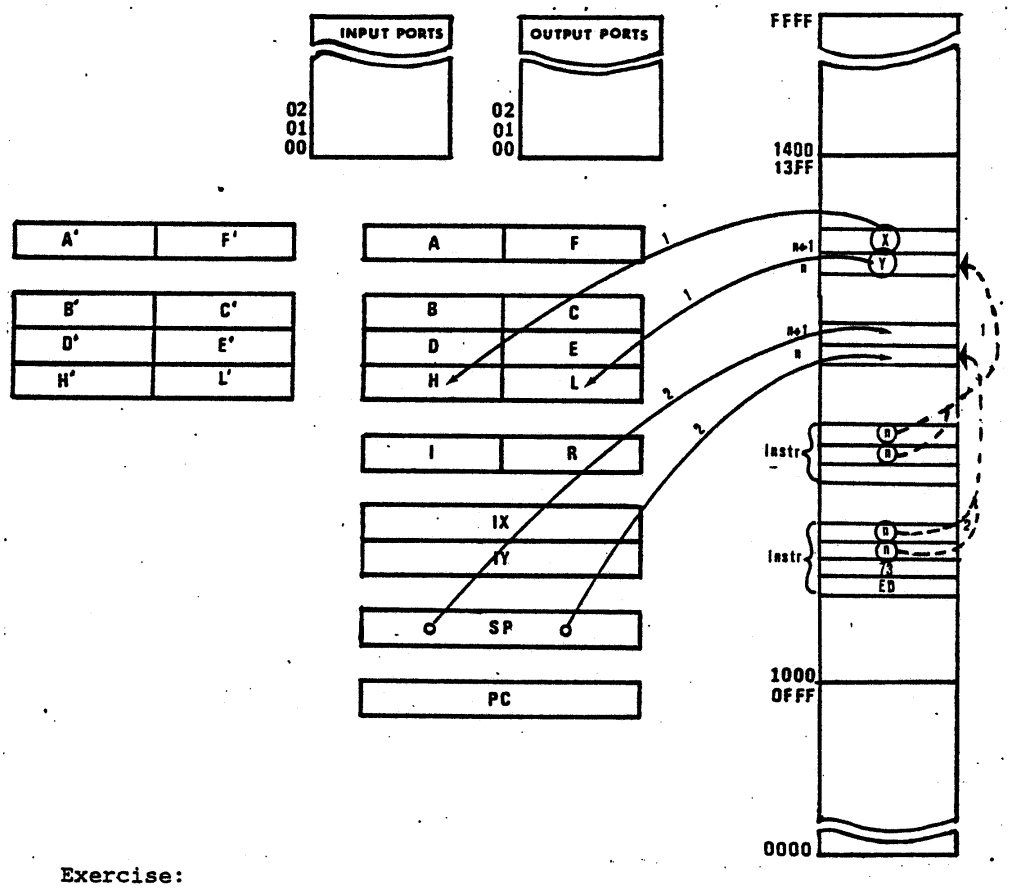
Exercise:

1. Initialize the B register to zero and the C register to contain ABH.

2. Cause the IX register to point to memory location page 11H line 32H.

16 Bit Memory Reference Load

1. LD HL (nn) 3 Byte Load from Memory
2. LD (nn) SP 4 Byte Load to Memory

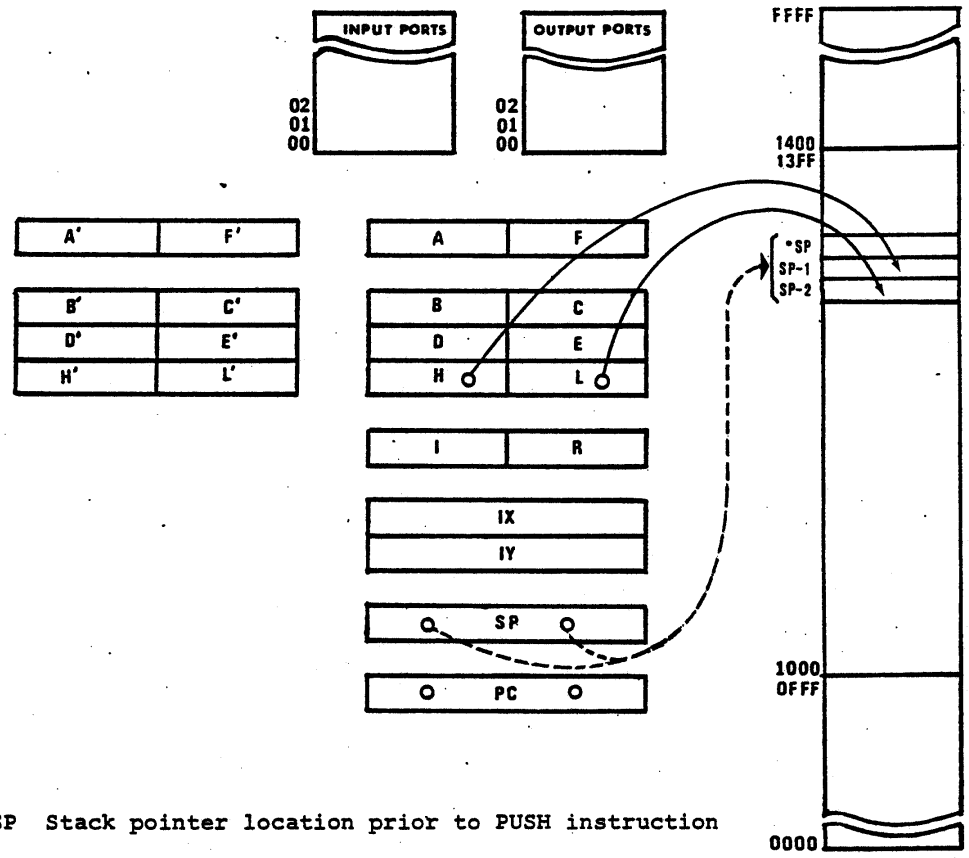


Exercise:

Transfer six consecutive bytes in a memory buffer starting on address Ø1BØ and ending on Ø1B5, into registers BCDEH and L preserving the order.

PUSH INSTRUCTION

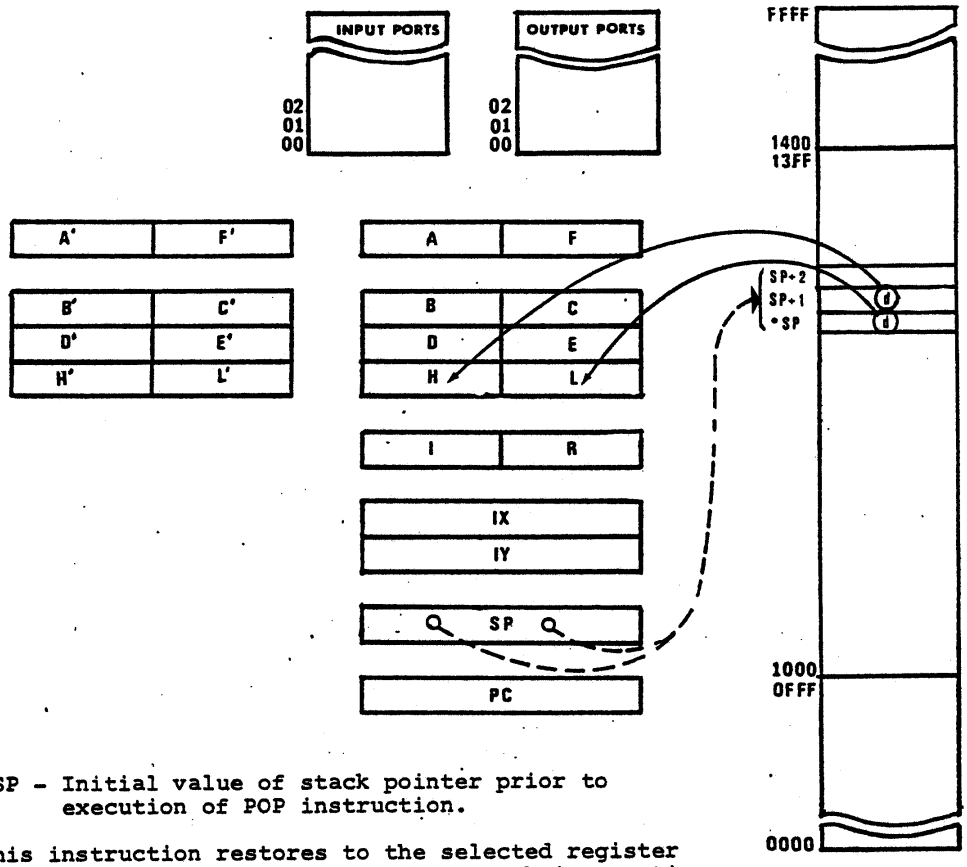
PUSH HL 1 Byte Register Pair Save Instruction



*SP Stack pointer location prior to PUSH instruction

This instruction saves the contents of the designated register pair in a portion of RAM Memory Space reserved for stack operations. The stack pointer is decremented automatically.

POP Instruction
POP HL 1 Byte Register Pair Restore Instruction



*SP - Initial value of stack pointer prior to execution of POP instruction.

This instruction restores to the selected register pair the data previously saved by a push instruction. the stack pointer is incremented automatically after each byte of data is returned.

There is no counterpart in the 8080 instruction set for the Exchange, Transfer and Search instructions of the Z-80*.

Exchange, Block Transfer and Search Instructions

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
EX DE, HL	DE ↔ HL	•	•	•	•	•	•	11 101 011	1	1	4	
EX AF, AF'	AF ↔ AF'	•	•	•	•	•	•	00 001 000	1	1	4	
EXX	(BC DE HL) ↔ (BC' DE' HL')	•	•	•	•	•	•	11 011 001	1	1	4	Register bank and auxiliary register bank exchange
EX (SP), HL	H ↔ (SP+1) L ↔ (SP)	•	•	•	•	•	•	11 100 011	1	5	19	
EX (SP), IX	IX_H ↔ (SP+1) IX_L ↔ (SP)	•	•	•	•	•	•	11 011 101 11 100 011	2	6	23	
EX (SP), IY	IY_H ↔ (SP+1) IY_L ↔ (SP)	•	•	•	•	•	•	11 111 101 11 100 011	2	6	23	
LDI	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1	•	•	① ↕	•	0	0	11 101 101 10 100 000	2	4	16	Load (HL) into (DE), increment the pointers and decrement the byte counter (BC)
LDIR	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1 Repeat until BC = 0	•	•	0	•	0	0	11 101 101 10 110 000	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
LDD	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1	•	•	① ↕	•	0	0	11 101 101 10 101 000	2	4	16	
LDDR	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1 Repeat until BC = 0	•	•	0	•	0	0	11 101 101 10 111 000	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
CPI	A - (HL) HL ← HL+1 BC ← BC-1	•	② ↕	① ↕	↕	1	↕	11 101 101 10 100 001	2	4	16	
CPIR	A - (HL) HL ← HL+1 BC ← BC-1 Repeat until A = (HL) or BC = 0	•	② ↕	① ↕	↕	1	↕	11 101 101 10 110 001	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)
CPD	A - (HL) HL ← HL-1 BC ← BC-1	•	② ↕	① ↕	↕	1	↕	11 101 101 10 101 001	2	4	16	
CPDR	A - (HL) HL ← HL-1 BC ← BC-1 Repeat until A = (HL) or BC = 0	•	② ↕	① ↕	↕	1	↕	11 101 101 10 111 001	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)

Notes:

① P/V flag is 0 if the result of BC-1 = 0, otherwise P/V = 1

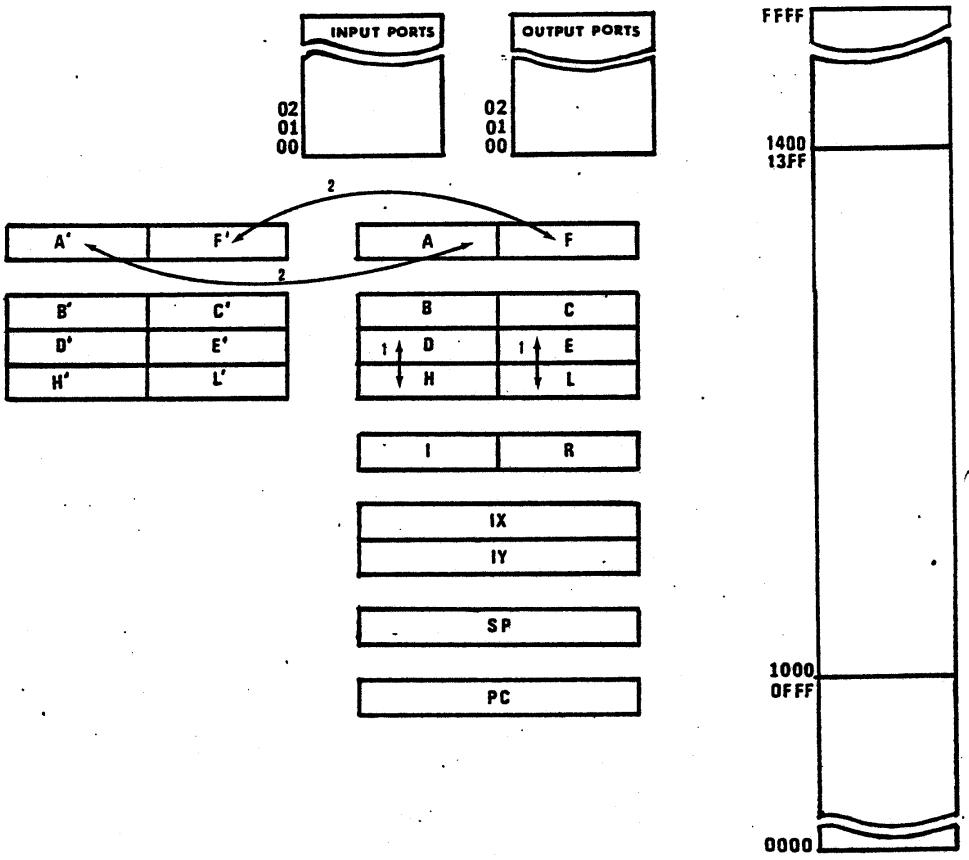
② Z flag is 1 if A = (HL), otherwise Z = 0.

*Except for EX DE, HL and EX(SP), HL

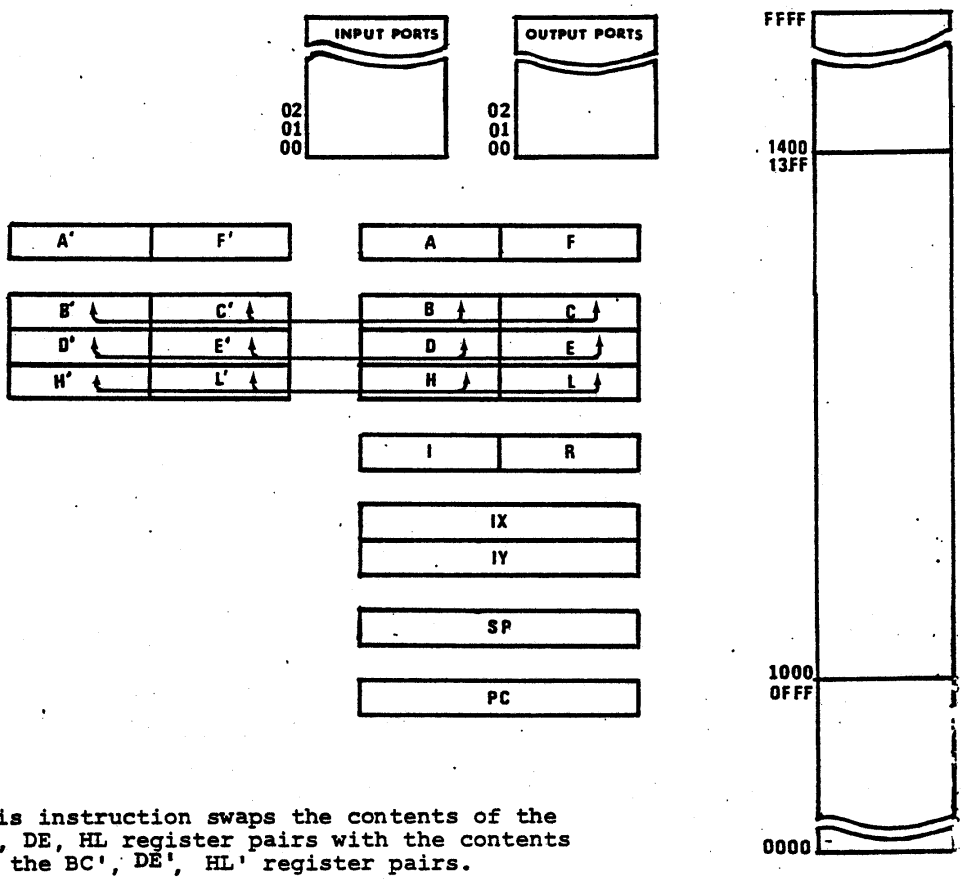
F

1 EX DE, HL
2 EX AF, AF'

1 Byte Exchange Instructions



EXX 1 Byte Working Register Block Exchange



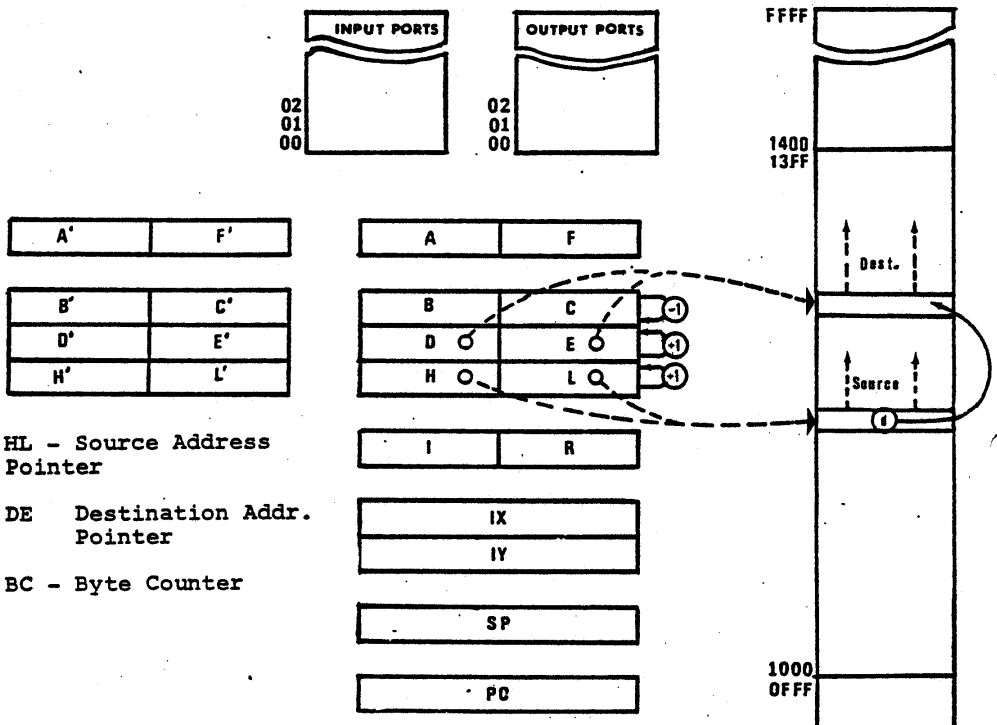
This instruction swaps the contents of the BC, DE, HL register pairs with the contents of the BC', DE', HL' register pairs.

Exercise:

Copy Registers DE, HL into DE' and HL' respectfully. Do not change either BC or BC'.

Transfer instructions are used to transfer a block of data from a source area in Memory Space to a destination area. This can be accomplished one byte at a time using the LDI instruction or continuously with the LDIR instruction.

LDI - A 2 Byte Transfer Instruction
LDIR - A 2 Byte Transfer and Repeat Instruction



HL - Source Address Pointer

DE Destination Addr. Pointer

BC - Byte Counter

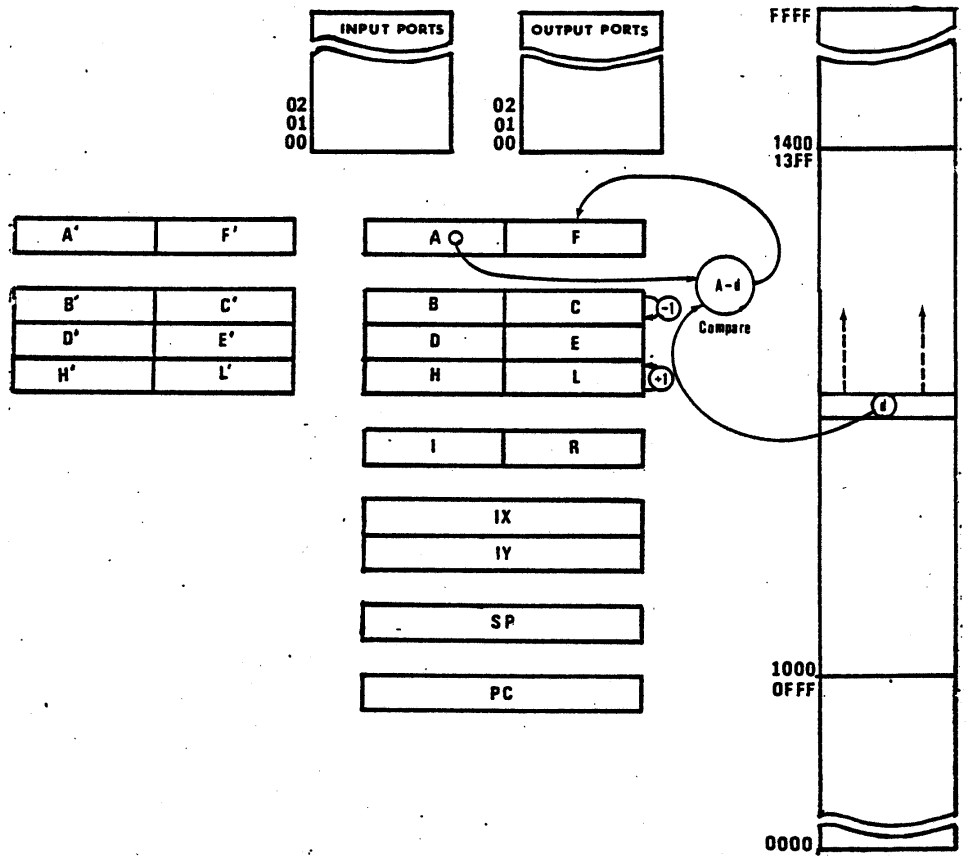
The LDI instruction will transfer a byte of data from a source address in memory space to a destination address each time the instruction is executed. When the decremented Byte Counter, BC-1 = 0 P/V is reset (P/V is set if BC-1 ≠ 0)

The LDIR instruction is similar to the LDI except execution of the instruction will be repeated automatically until BC-1 = 0.

Exercise: Write mnemonics to transfer $(256)_{10}$ bytes from page Ø to page 1Ø.

Search Instructions

CPI - 2 Byte Compare and Increment Instruction
CPIR - 2 Byte Compare, Increment and Repeat Instruction



The CPI instruction compares the contents of a memory location specified by the HL register pair with the contents of the Accumulator. The HL pointer is incremented, and the Byte Counter BC is decremented. If A = (HL), Z is set. otherwise it is reset. P/V is set if BC-1 ≠ 0, otherwise it is reset.

The CPIR instruction is the same as the CPI, except execution of the instruction is repeated untill a match is found, or untill BC-1 = 0, at which time the instruction is terminated.

8-Bit Arithmetic and Logical Operations

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
ADD r	A ← A + r	‡	‡	V	‡	0	‡	10 [000] r	1	1	4	r Reg. 000 B 001 C 010 D
ADD n	A ← A + n	‡	‡	V	‡	0	‡	11 [000] 110 ← n →	2	2	7	
ADD (HL)	A ← A + (HL)	‡	‡	V	‡	0	‡	10 [000] 110	1	2	7	011 E 100 H 101 L 111 A
ADD (IX+d)	A←A + (IX+d)	‡	‡	V	‡	0	‡	11 011 101 10 [000] 110 ← d →	3	5	19	
ADD (IY+d)	A←A+(IY+d)	‡	‡	V	‡	0	‡	11 111 101 10 [000] 110 ← d →	3	5	19	
ADC s	A ← A + s + CY	‡	‡	V	‡	0	‡	[001]				s is any of r, n, (HL), (IX+d), (IY+d) as shown for ADD instruction The indicated bits replace the 000 in the ADD set above.
SUB s	A ← A − s	‡	‡	V	‡	1	‡	[010]				
SBC s	A ← A − s − CY	‡	‡	V	‡	1	‡	[011]				
AND s	A ← A ∧ s	0	‡	P	‡	0	‡	[100]				
OR s	A ← A ∨ s	0	‡	P	‡	0	‡	[110]				
XOR s	A ← A ∘ s	0	‡	P	‡	0	‡	[101]				
CP s	A − s	‡	‡	V	‡	1	‡	[111]				
INC r	r ← r + 1	•	‡	V	‡	0	‡	00 r [100]	1	1	4	
INC (HL)	(HL) ← (HL)+1	•	‡	V	‡	0	‡	00 110 [100]	1	3	11	
INC (IX+d)	(IX+d) ← (IX+d)+1	•	‡	V	‡	0	‡	11 011 101 00 110 [100] ← d →	3	6	23	
INC (IY+d)	(IY+d) ← (IY+d) + 1	•	‡	V	‡	0	‡	11 111 101 00 110 [100] ← d →	3	6	23	
DEC d	d ← d − 1	•	‡	V	‡	1	‡	[101]				d is any of r, (HL), (IX+d), (IY+d) as shown for INC. Same format and states as INC. Replace 100 with 101 in OP code.

Notes: The V symbol in the P/V flag column indicates that the P/V flag contains the overflow of the result of the operation. Similarly the P symbol indicates parity. V = 1 means overflow, V = 0 means not overflow. P = 1 means parity of the result is even, P = 0 means parity of the result is odd.

Flag Notation: • = flag not affected. 0 = flag reset, 1 = flag set, X = flag is unknown.
‡ = flag is affected according to the result of the operation.

Courtesy Zilog Corp.

BIT SET, RESET AND TEST

BIT OPS	BIT	REGISTER ADDRESSING							REG. INDIR.	INDEXED	
		A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
TEST "BIT"	0	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
	1	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
	2	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
	3	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
	4	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
	5	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
	6	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
	7	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
RESET BIT "RES"	0	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
	1	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
	2	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
	3	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
	4	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
	5	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
	6	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
	7	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET BIT "SET"	0	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
	1	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
	2	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
	3	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
	4	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
	5	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
	6	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
	7	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

Courtesy Zilog Corp.

F

General Purpose Arithmetic and CPU Control Instructions

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
DAA	Converts acc. content into packed BCD following add or subtract with packed BCD operands	‡	‡	P	‡	•	‡	00 100 111	1	1	4	Decimal adjust accumulator
CPL	$A \leftarrow \overline{A}$	•	•	•	•	1	1	00 101 111	1	1	4	Complement accumulator (one's complement)
NEG	$A \leftarrow 0 - A$	‡	‡	V	‡	1	‡	11 101 101 01 000 100	2	2	8	Negate acc. (two's complement)
CCF	$CY \leftarrow \overline{CY}$	‡	•	•	•	0	X	00 111 111	1	1	4	Complement carry flag
SCF	CY ← 1	1	•	•	•	0	0	00 110 111	1	1	4	Set carry flag
NOP	No operation	•	•	•	•	•	•	00 000 000	1	1	4	
HALT	CPU halted	•	•	•	•	•	•	01 110 110	1	1	4	
DI	IFF ← 0	•	•	•	•	•	•	11 110 011	1	1	4	
EI	IFF ← 1	•	•	•	•	•	•	11 111 011	1	1	4	
IM 0	Set interrupt mode 0	•	•	•	•	•	•	11 101 101 01 000 110	2	2	8	
IM 1	Set interrupt mode 1	•	•	•	•	•	•	11 101 101 01 010 110	2	2	8	
IM2	Set interrupt mode 2	•	•	•	•	•	•	11 101 101 01 011 110	2	2	8	

Notes: IFF indicates the interrupt enable flip-flop
CY indicates the carry flip-flop.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

Courtesy Zilog Corp.

16 Bit Arithmetic Operations

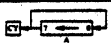
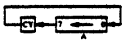
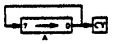
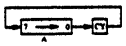
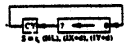
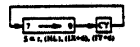
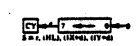
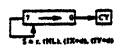
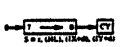
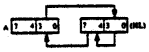
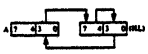
Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
ADD HL, ss	HL ← HL+ss	‡	•	•	•	0	X	00 ss1 001	1	3	11	ss Reg. 00 BC 01 DE 10 HL 11 SP
ADC HL, ss	HL←HL+ss+CY	‡	‡	V	‡	0	X	11 101 101 01 ss1 010	2	4	15	
SBC HL, ss	HL←HL-ss-CY	‡	‡	V	‡	1	X	11 101 101 01 ss0 010	2	4	15	
ADD IX, pp	IX ← IX + pp	‡	•	•	•	0	X	11 011 101 00 ppi 001	2	4	15	pp Reg. 00 BC 01 DE 10 IX 11 SP
ADD IY, rr	IY←IY+rr	‡	•	•	•	0	X	11 111 101 00 rr1 001	2	4	15	rr Reg. 00 BC 01 DE 10 IY 11 SP
INC ss	ss ← ss + 1	•	•	•	•	•	•	00 ss0 011	1	1	6	
INC IX	IX ← IX + 1	•	•	•	•	•	•	11 011 101 00 100 011	2	2	10	
INC IY	IY ← IY + 1	•	•	•	•	•	•	11 111 101 00 100 011	2	2	10	
DEC ss	ss ← ss - 1	•	•	•	•	•	•	00 ss1 011	1	1	6	
DEC IX	IX ← IX - 1	•	•	•	•	•	•	11 011 101 00 101 011	2	2	10	
DEC IY	IY ← IY - 1	•	•	•	•	•	•	11 111 101 00 101 011	2	2	10	

Notes: ss is any of the register pairs BC, DE, HL, SP
pp is any of the register pairs BC, DE, IX, SP
rr is any of the register pairs BC, DE, IY, SP.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
‡ = flag is affected according to the result of the operation.

Courtesy Zilog Corp.

Rotate and Shift Operations

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
RLCA		‡	•	•	•	0	0	00 000 111	1	1	4	Rotate left circular accumulator
RLA		‡	•	•	•	0	0	00 010 111	1	1	4	Rotate left accumulator
RRCA		‡	•	•	•	0	0	00 001 111	1	1	4	Rotate right circular accumulator
RRA		‡	•	•	•	0	0	00 011 111	1	1	4	Rotate right accumulator
RLC r		‡	‡	P	‡	0	0	11 001 011 00 000 r	2	2	8	Rotate left circular register r
RLC (HL)		‡	‡	P	‡	0	0	11 001 011 00 000 110	2	4	15	r Reg. 000 B 001 C 010 D 011 E 100 H 101 L 111 A
RLC (IX+d)		‡	‡	P	‡	0	0	11 011 101 11 001 011 ← d → 00 000 110	4	6	23	
RLC (IY+d)		‡	‡	P	‡	0	0	11 111 101 11 001 011 ← d → 00 000 110	4	6	23	
RL s		‡	‡	P	‡	0	0	010				Instruction format and states are as shown for RLC,s. To form new OP-code replace 000 of RLC,s with shown code
RRC s		‡	‡	P	‡	0	0	001				
RR s		‡	‡	P	‡	0	0	011				
SLA s		‡	‡	P	‡	0	0	100				
SRA s		‡	‡	P	‡	0	0	101				
SRL s		‡	‡	P	‡	0	0	111				
RLD		•	‡	P	‡	0	0	11 101 101 01 101 111	2	5	18	Rotate digit left and right between the accumulator and location (HL). The content of the upper half of the accumulator is unaffected
RRD		•	‡	P	‡	0	0	11 101 101 01 100 111	2	5	18	

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

Courtesy Zilog Corp.

Bit Set, Reset and Test Instructions

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
BIT b, r	$Z \leftarrow \overline{r_b}$	•	‡	X	X	0	1	11 001 011 01 b r	2	2	8	r: 000 B, 001 C, 010 D, 011 E, 100 H, 101 L, 111 A
BIT b, (HL)	$Z \leftarrow \overline{(HL)_b}$	•	‡	X	X	0	1	11 001 011 01 b 110	2	3	12	b: 000 Bit 0, 001 Bit 1, 010 Bit 2, 011 Bit 3, 100 Bit 4, 101 Bit 5, 110 Bit 6, 111 Bit 7
BIT b, (IX+d)	$Z \leftarrow \overline{(IX+d)_b}$	•	‡	X	X	0	1	11 011 101 11 001 011 ← d → 01 b 110	4	5	20	
BIT b, (IY+d)	$Z \leftarrow \overline{(IY+d)_b}$	•	‡	X	X	0	1	11 111 101 11 001 011 ← d → 01 b 110	4	5	20	
SET b, r	$r_b \leftarrow 1$	•	•	•	•	•	•	11 001 011 [11] b r	2	2	8	
SET b, (HL)	$(HL)_b \leftarrow 1$	•	•	•	•	•	•	11 001 011 [11] b 110	2	4	15	
SET b, (IX+d)	$(IX+d)_b \leftarrow 1$	•	•	•	•	•	•	11 011 101 11 001 011 ← d → [11] b 110	4	6	23	
SET b, (IY+d)	$(IY+d)_b \leftarrow 1$	•	•	•	•	•	•	11 111 101 11 001 011 ← d → [11] b 110	4	6	23	
RES b, s	$s_b \leftarrow 0$ s ≡ r, (HL), (IX+d), (IY+d)							[10]				To form new OP-code replace [11] of SET b,s with [10]. Flags and time states for SET instruction

Notes: The notation s_b indicates bit b (0 to 7) of location s.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
‡ = flag is affected according to the result of the operation.

Courtesy Zilog Corp.

Jump Instructions

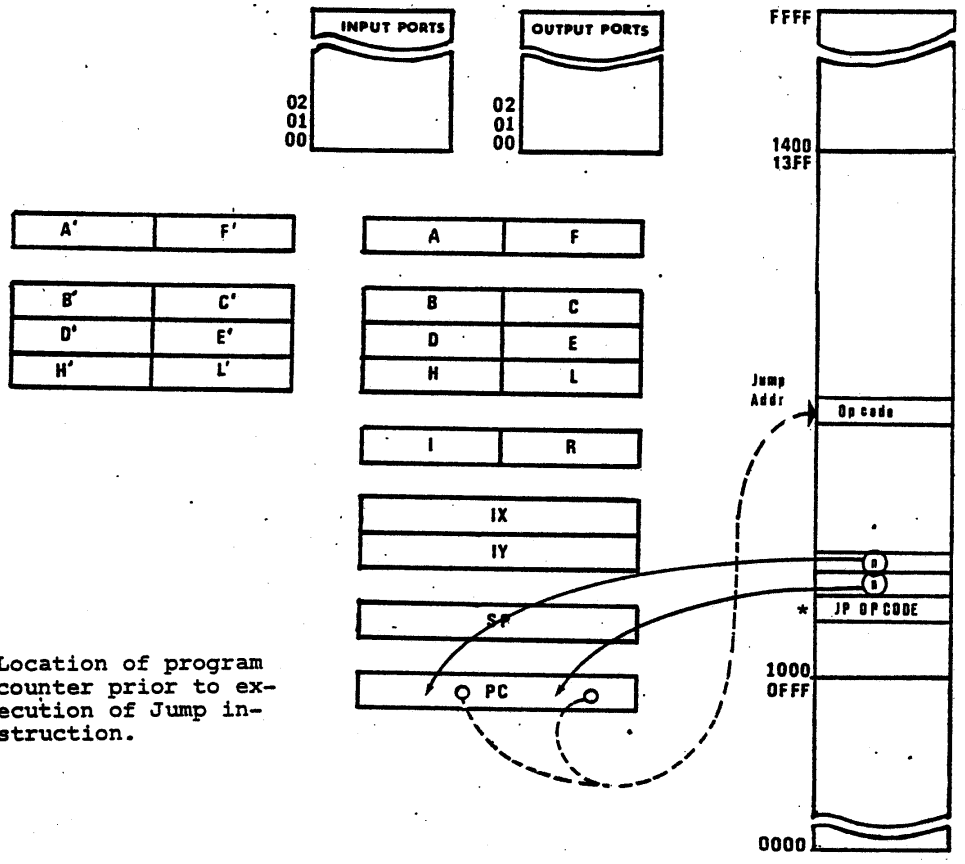
Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
JP nn	PC ← nn	•	•	•	•	•	•	11 000 011 ← n → ← n →	3	3	10	
JP cc, nn	If condition cc is true PC ← nn, otherwise continue	•	•	•	•	•	•	11 cc 010 ← n → ← n →	3	3	10	cc: 000 NZ non zero; 001 Z zero; 010 NC non carry; 011 C carry; 100 PO parity odd; 101 PE parity even; 110 P sign positive; 111 M sign negative
JR e	PC ← PC + e	•	•	•	•	•	•	00 011 000 ← e-2 →	2	3	12	
JR C, e	If C = 0, continue	•	•	•	•	•	•	00 111 000 ← e-2 →	2	2	7	If condition not met
	If C = 1, PC ← PC+e								2	3	12	If condition is met
JR NC, e	If C = 1, continue	•	•	•	•	•	•	00 110 000 ← e-2 →	2	2	7	If condition not met
	If C = 0, PC ← PC + e								2	3	12	If condition is met
JR Z, e	If Z = 0 continue	•	•	•	•	•	•	00 101 000 ← e-2 →	2	2	7	If condition not met
	If Z = 1, PC ← PC + e								2	3	12	If condition is met
JR NZ, e	If Z = 1, continue	•	•	•	•	•	•	00 100 000 ← e-2 →	2	2	7	If condition not me
	If Z = 0, PC ← PC + e								2	3	12	If condition met
JP (HL)	PC ← HL	•	•	•	•	•	•	11 101 001	1	1	4	
JP (IX)	PC ← IX	•	•	•	•	•	•	11 011 101 11 101 001	2	2	8	
JP (IY)	PC ← IY	•	•	•	•	•	•	11 111 101 11 101 001	2	2	8	
DJNZ,e	B ← B-1 If B = 0, continue	•	•	•	•	•	•	00 010 000 ← e-2 →	2	2	8	If B = 0
	If B ≠ 0, PC ← PC + e								2	3	13	IF B ≠ 0

Notes: e represents the extension in the relative addressing mode.
e is a signed two's complement number in the range <-126, 129>
e-2 in the op-code provides an effective address of pc +e as PC is incremented by 2 prior to the addition of e.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

Jump Instructions

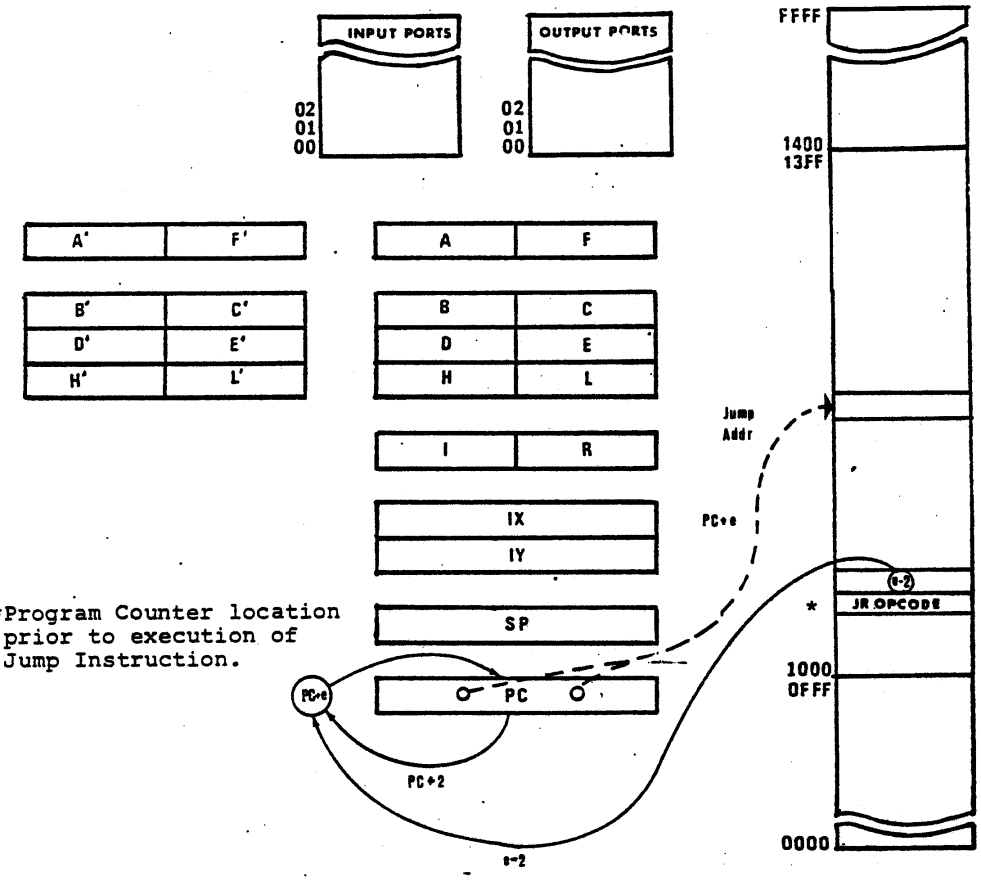
1. JP nn — 3 Byte Unconditional Jump to nn.
2. JP cc, nn - 3 Byte Conditional Jump to nn.



*Location of program counter prior to execution of Jump instruction.

The unconditional Jump instruction JP nn advances the Program Counter to the Jump Address nn. Program execution proceeds from this point. The conditional Jump instruction requires a specified flag condition cc be met to initiate the Jump; otherwise program execution proceeds with next instruction after the Jump instruction.

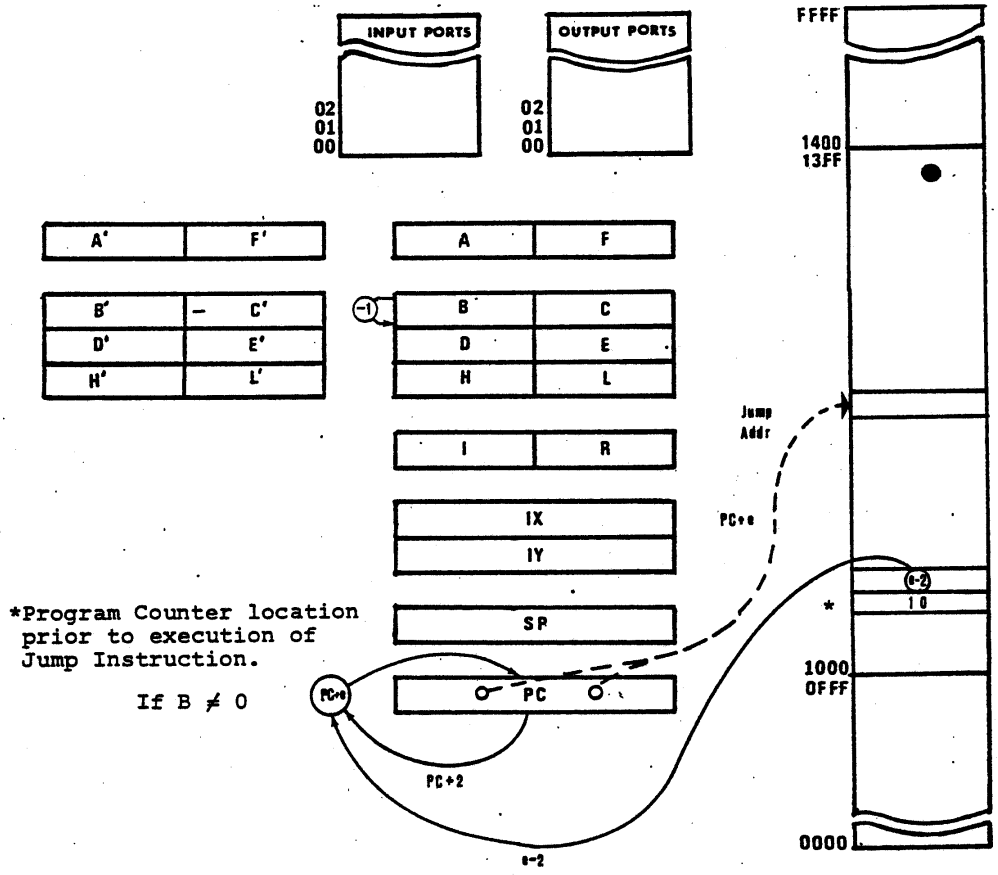
JR e 2-Byte Unconditional Relative Jump Instruction
JR cc, e 2-Byte Conditional Relative Jump Instruction



*Program Counter location prior to execution of Jump Instruction.

In the JR (relative jump) instructions, the value of the displacement e is added to the Program Counter; the next instruction is then selected from the location specified by PC+e. If the jump instruction is conditional, however, the next adjacent instruction will be executed unless the condition specified is true.

DJNZ, e 2-Byte Conditional Decrement and Jump Instruction



*Program Counter location prior to execution of Jump Instruction.

If B ≠ 0

In the DJNZ instruction, the B register is used to determine branching. The B register is decremented and if a non-zero value remains, the value of the displacement e is added to the Program Counter; the next instruction is selected from location specified by PC+e. If B=0 after decrementing the next adjacent instruction is executed.

Call and Return Instructions

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
CALL nn	$(SP-1)\leftarrow PC_H$ $(SP-2)\leftarrow PC_L$ $PC\leftarrow nn$	•	•	•	•	•	•	11 001 101 ← n → ← n →	3	5	17	
CALL cc, nn	If condition cc is false continue, otherwise same as CALL nn	•	•	•	•	•	•	11 cc 100 ← n → ← n →	3 3	3 5	10 17	If cc is false If cc is true
RET	$PC_L\leftarrow(SP)$ $PC_H\leftarrow(SP+1)$	•	•	•	•	•	•	11 001 001	1	3	10	
RET cc	If condition cc is false continue, otherwise same as RET	•	•	•	•	•	•	11 cc 000	1 1	1 3	5 11	If cc is false If cc is true
RETI	Return from interrupt	•	•	•	•	•	•	11 101 101 01 001 101	2	4	14	
RETN	Return from non maskable interrupt	•	•	•	•	•	•	11 101 101 01 000 101	2	4	14	
RST p	$(SP-1)\leftarrow PC_H$ $(SP-2)\leftarrow PC_L$ $PC_H\leftarrow 0$ $PC_L\leftarrow P$	•	•	•	•	•	•	11 t 111	1	3	11	

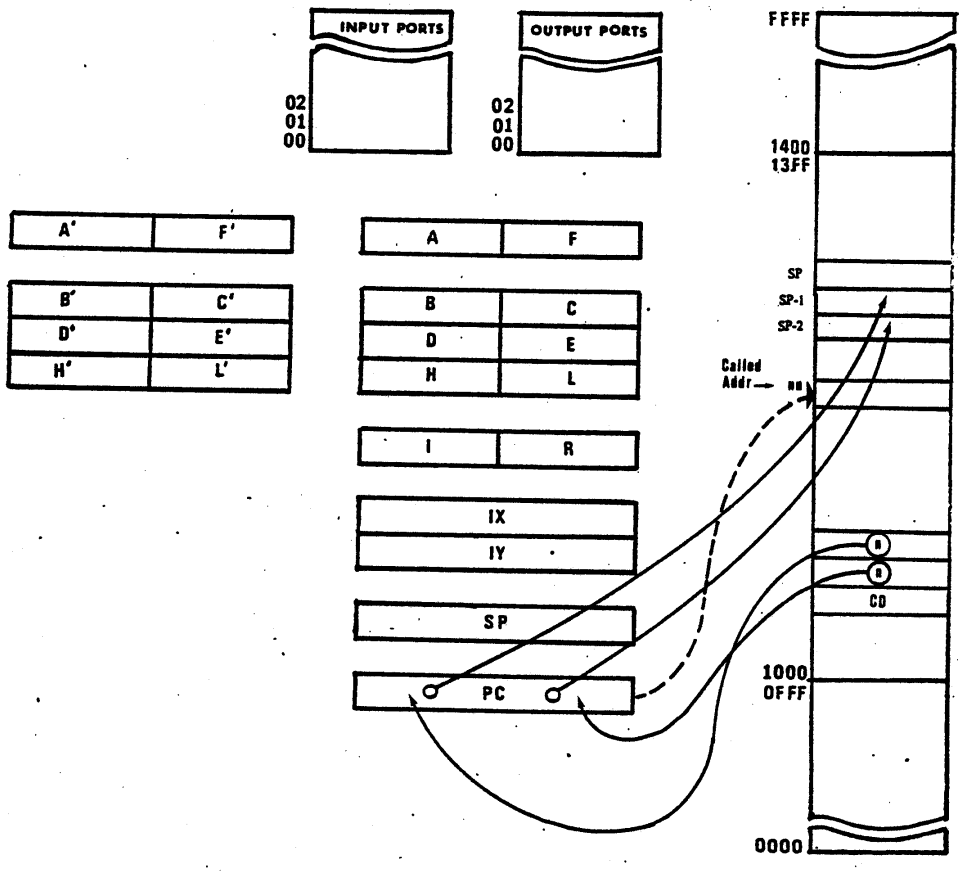
cc	Condition	
000	NZ	non zero
001	Z	zero
010	NC	non carry
011	C	carry
100	PO	parity odd
101	PE	parity even
110	P	sign positive
111	M	sign negative

t	P
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown
‡ = flag is affected according to the result of the operation.

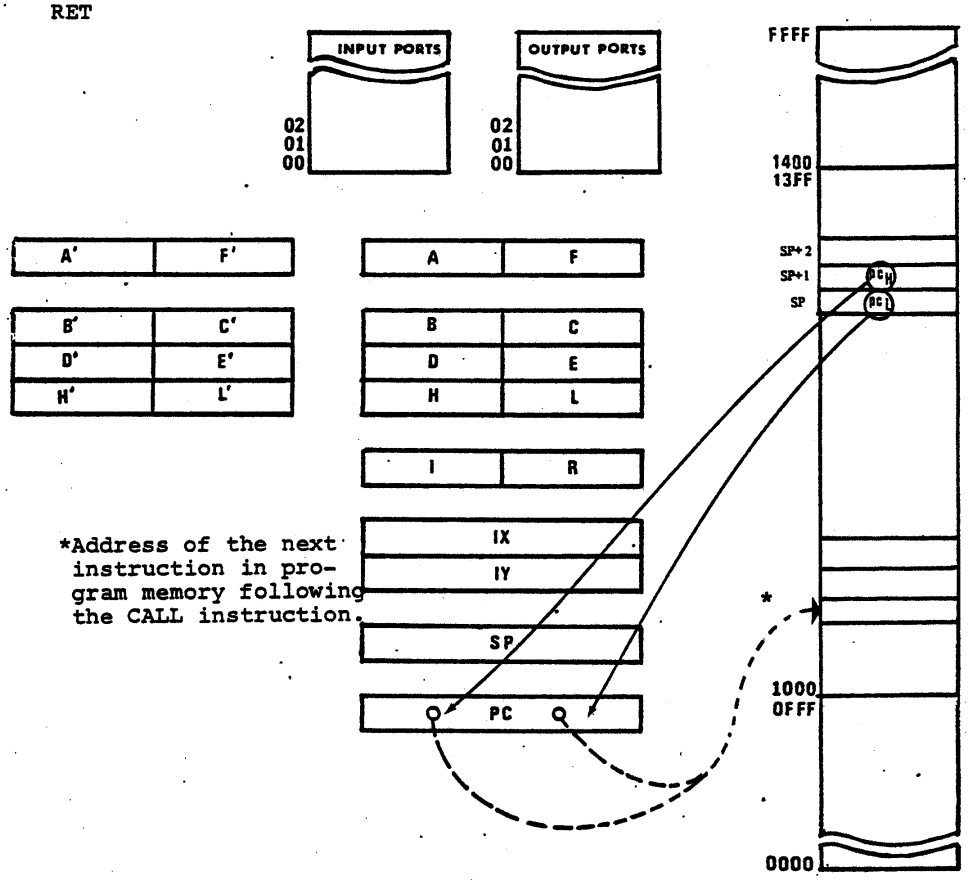
Courtesy Zilog Corporation

CALL nn



CALL nn is used to call a subroutine. Execution of the instruction stores the contents of the program counter in the stack, then jumps to the location of the subroutine specified by nn.

RET



This one byte instruction returns the address of the next instruction following the CALL to the Program Counter. Execution of main program is resumed.

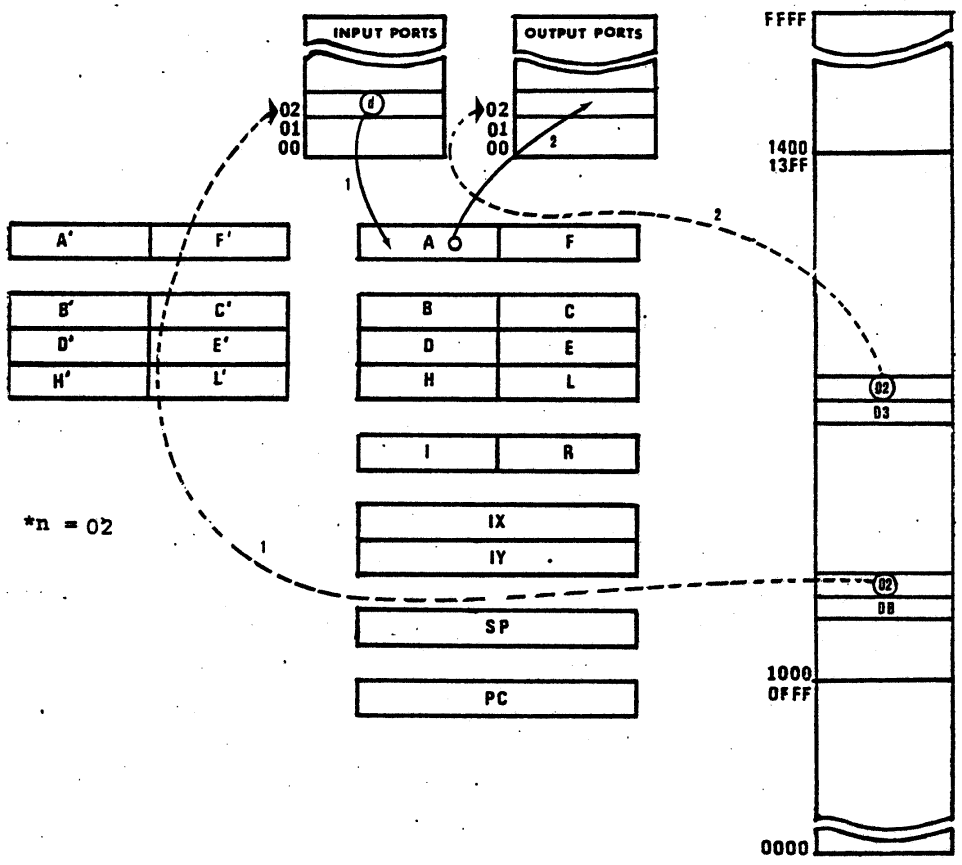
Input/Output Instructions

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
IN A, (n)	A ← (n)	•	•	•	•	•	•	11 011 011 ← n →	2	3	10	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
IN r, (C)	r ← (C) if r = 110 only the flags will be affected	•	‡	P	‡	0	‡	11 101 101 01 r 000	2	3	11	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INI	(HL) ← (C) B ← B - 1 HL ← HL + 1	•	‡ ①	X	X	1	X	11 101 101 10 100 010	2	4	15	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INIR	(HL) ← (C) B ← B - 1 HL ← HL + 1 Repeat until B = 0	•	1	X	X	1	X	11 101 101 10 110 010	2 2	5 (If B ≠ 0) 4 (If B = 0)	20 15	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
IND	(HL) ← (C) B ← B - 1 HL ← HL - 1	•	‡ ①	X	X	1	X	11 101 101 10 101 010	2	4	15	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INDR	(HL) ← (C) B ← B - 1 HL ← HL -1 Repeat until B = 0	•	1	X	X	1	X	11 101 101 10 111 010	2 2	5 (If B ≠ 0) 4 (If B = 0)	20 15	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUT (n), A	(n) ← A	•	•	•	•	•	•	11 010 011	2	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
OUT (C), r	(C) ← r	•	•	•	•	•	•	11 101 101 01 r 001	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTI	(C) ← (HL) B ← B - 1 HL ← HL + 1	•	‡ ①	X	X	1	X	11 101 101 10 100 011	2	4	15	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTIR	(C) ← (HL) B ← B - 1 HL ← HL + 1 Repeat until B = 0	•	1	X	X	1	X	11 101 101 10 110 011	2 2	5 (If B ≠ 0) 4 (If B = 0)	20 15	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTD	(C) ← (HL) B ← B - 1 HL ← HL - 1	•	‡ ①	X	X	1	X	11 101 101 10 101 011	2	4	15	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTDR	(C) ← (HL) B ← B - 1 HL ← HL -1 Repeat until B = 0	•	1	X	X	1	X	11 101 101 10 111 011	2 2	5 (If B ≠ 0) 4 (If B = 0)	20 15	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$

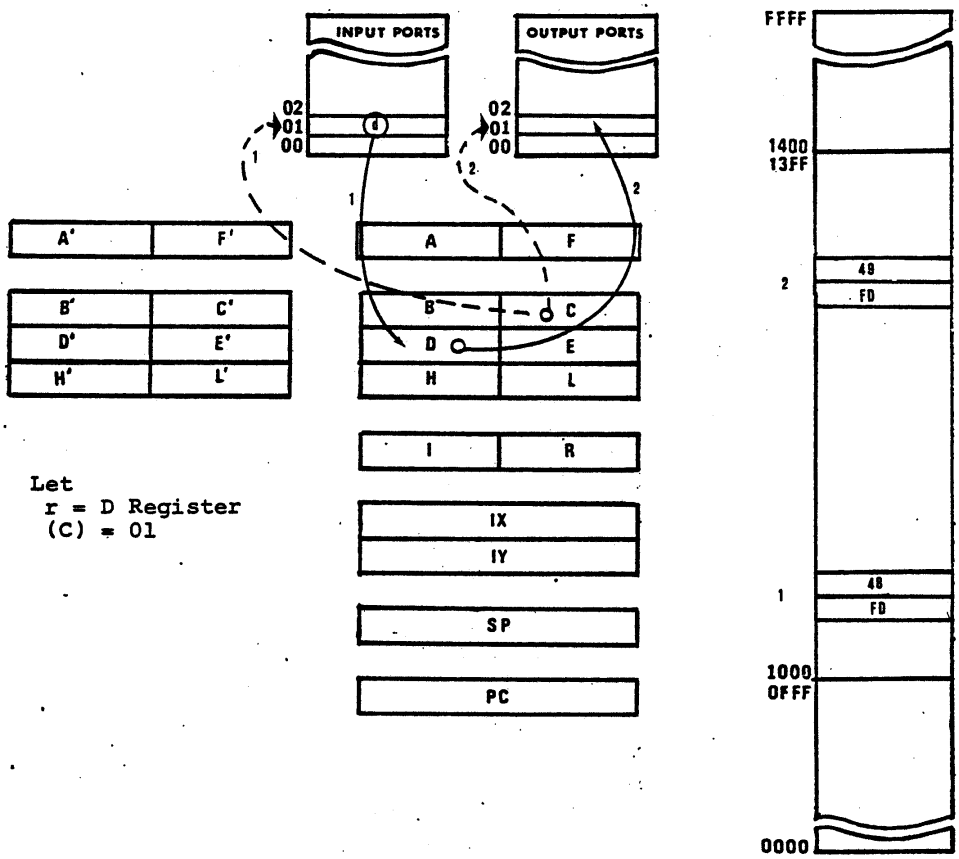
Notes: ① If the result of B - 1 is zero the Z flag is set, otherwise it is reset.

Courtesy of Zilog Corp.

1. IN A (n) - 2 Byte Input Instruction
2. OUT (n) A - 2 Byte Output Instruction



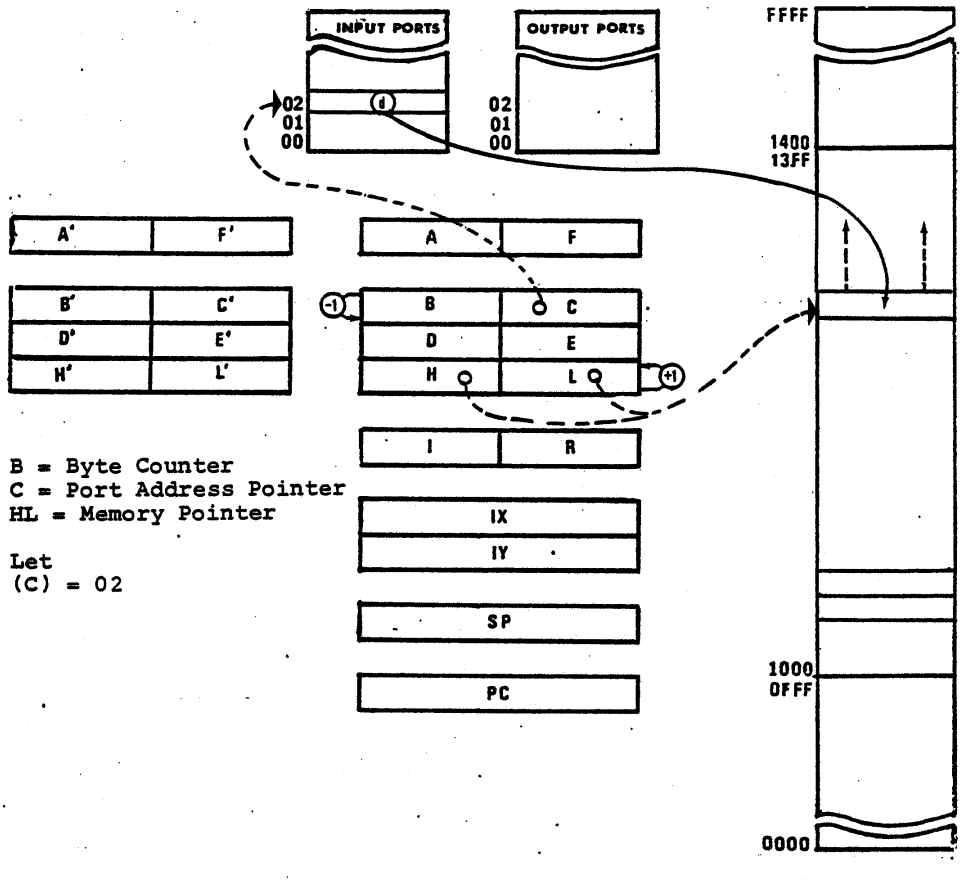
1. IN r (C) - 2 Byte Input Instruction (Indirect Address)
2. OUT (C) r - 2 Byte Output Instruction (Indirect Address)



Let
r = D Register
(C) = 01

In this instruction the C Register is used as an Address Pointer for port selection.

1. INI 2 Byte Input and Increment Instruction
2. INIR 2 Byte Input, Increment and Repeat Instruction



DECIMAL COUNT	CONVERSION FRWD	CONVERSION BKWD	DECIMAL COUNT	CONVERSION FRWD	CONVERSION BKWD	DECIMAL COUNT	CONVERSION FRWD	CONVERSION BKWD
000	--	--						
001	01	FF	049	31	CF	097	61	9F
002	02	FE	050	32	CE	098	62	9E
003	03	FD	051	33	CD	099	63	9D
004	04	FC	052	34	CC	100	64	9C
005	05	FB	053	35	CB	101	65	9B
006	06	FA	054	36	CA	102	66	9A
007	07	F9	055	37	C9	103	67	99
008	08	F8	056	38	C8	104	68	98
009	09	F7	057	39	C7	105	69	97
010	0A	F6	058	3A	C6	106	6A	96
011	0B	F5	059	3B	C5	107	6B	95
012	0C	F4	060	3C	C4	108	6C	94
013	0D	F3	061	3D	C3	109	6D	93
014	0E	F2	062	3E	C2	110	6E	92
015	0F	F1	063	3F	C1	111	6F	91
016	10	F0	064	40	C0	112	70	90
017	11	EF	065	41	BF	113	71	8F
018	12	EE	066	42	BE	114	72	8E
019	13	ED	067	43	BD	115	73	8D
020	14	EC	068	44	BC	116	74	8C
021	15	EB	069	45	BB	117	75	8B
022	16	EA	070	46	BA	118	76	8A
023	17	E9	071	47	B9	119	77	89
024	18	E8	072	48	B8	120	78	88
025	19	E7	073	49	B7	121	79	87
026	1A	E6	074	4A	B6	122	7A	86
027	1B	E5	075	4B	B5	123	7B	85
028	1C	E4	076	4C	B4	124	7C	84
029	1D	E3	077	4D	B3	125	7D	83
030	1E	E2	078	4E	B2	126	7E	82
031	1F	E1	079	4F	B1	127	7F	81
032	20	E0	080	50	B0	128		80
033	21	DF	081	51	AF			
034	22	DE	082	52	AE			
035	23	DD	083	53	AD			
036	24	DC	084	54	AC			
037	25	DB	085	55	AB			
038	26	DA	086	56	AA			
039	27	D9	087	57	A9			
040	28	D8	088	58	A8			
041	29	D7	089	59	A7			
042	2A	D6	090	5A	A6			
043	2B	D5	091	5B	A5			
044	2C	D4	092	5C	A4			
045	2D	D3	093	5D	A3			
046	2E	D2	094	5E	A2			
047	2F	D1	094	5F	A1			
048	30	D0	096	60	A0			

BRANCH ADDRESS CONVERSION CHART

ADR	INSTR	LABEL	OPERATION	OPERAND	COMMENTS	BY:
0						
1						
2						
3						
4						
5						
6						
7						
8						
9						
A						
B						
C						
D						
E						
F						

ADR	
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	A
B	B
C	C
D	D
E	E
F	F

SECTION ZPR

Programming Instruction Set ZPR

Z80-CPU INSTRUCTION SET

INSTRUCTION	C	Z	P/V	S	N	H	COMMENTS
ADD A, s; ADC A, s	↕	↕	V	↕	0	↕	8-bit add or add with carry
SUB s; SBC A, s; CP s; NEG	↕	↕	V	↕	1	↕	8-bit subtract, subtract with carry, compare and negate accumulator
AND s	0	↕	P	↕	0	1	Logical operations
OR s; XOR s	0	↕	P	↕	0	0	And sets different flags
INC s	•	↕	V	↕	0	↕	8-bit increment
DEC m	•	↕	V	↕	1	↕	8-bit decrement
ADD DD, ss	↕	•	•	•	0	X	16-bit add
ADC HL, ss	↕	↕	V	↕	0	X	16-bit add with carry
SBC HL, ss	↕	↕	V	↕	1	X	16-bit subtract with carry
RLA; RLCA, RRA, RRCA	↕	•	•	•	0	0	Rotate accumulator
RL m; RLC m; RR m; RRC m SLA m; SRA m; SRL m	↕	↕	P	↕	0	0	Rotate and shift location s
RLD, RRD	•	↕	P	↕	0	0	Rotate digit left and right
DAA	↕	↕	P	↕	•	↕	Decimal adjust accumulator
CPL	•	•	•	•	1	1	Complement accumulator
SCF	1	•	•	•	0	0	Set carry
CCF	↕	•	•	•	0	X	Complement carry
IN r, (C)	•	↕	P	↕	0	0	Input register indirect
INI; IND; OUTI; OUTD	•	↕	X	X	1	X	Block input and output
INIR; INDR; OTIR; OTDR	•	1	X	X	1	X	Z = 0 if B ≠ 0 otherwise Z = 1
LDI, LDD	•	X	↕	X	0	0	Block transfer instructions
LDIR, LDDR	•	X	0	X	0	0	P/V = 1 if BC ≠ 0, otherwise P/V = 0
CPI, CPIR, CPD, CPDR	•	↕	↕	X	1	X	Block search instructions Z = 1 if A = (HL), otherwise Z = 0 P/V = 1 if BC = 0, otherwise P/V = 0
LD A, I; LD A, R	•	↕	IFF	↕	0	0	The content of the interrupt enable flip-flop (IFF) is copied into the P/V flag
BIT b, s	•	↕	X	X	0	1	The complement of bit b of location is copied into the Z flag
NEG	↕	↕	V	↕	1	↕	Negate accumulator

• The following notation is used in this table:

SYMBOL	OPERATION
C	Carry/link flag. C=1 if the operation produced a carry from the MSB of the operand or result.
Z	Zero flag. Z=1 if the result of the operation is zero.
S	Sign flag. S=1 if the MSB of the result is one.
P/V	Parity or overflow flag. Parity (P) and overflow (V) share the same flag. Logical operations affect this flag with the parity of the result while arithmetic operations affect this flag with the overflow of the result. If P/V holds parity, P/V=1 if the result of the operation is even, P/V=0 if result is odd. If P/V holds overflow, P/V=1 if the result of the operation produced an overflow.
H	Half-carry flag. H=1 if the add or subtract operation produced a carry into or borrow from bit 4 of the accumulator.
N	Add/Subtract flag. N=1 if the previous operation was a subtract.
	H and N flags are used in conjunction with the decimal adjust instruction (DAA) to properly correct the result into packed BCD format following addition or subtraction using operands with packed BCD format.
↕	The flag is affected according to the result of the operation.
•	The flag is unchanged by the operation.
0	The flag is reset by the operation.
1	The flag is set by the operation.
X	The flag is a "don't care."
V	P/V flag affected according to the overflow result of the operation.
P	P/V flag affected according to the parity result of the operation.
r	Any one of the CPU registers A, B, C, D, E, H, L.
s	Any 8-bit location for all the addressing modes allowed for the particular instruction.
ss	Any 16-bit location for all the addressing modes allowed for that instruction.
ii	Any one of the two index registers IX or IY.
R	Refresh counter.
n	8-bit value in range <0, 255>.
nn	16-bit value in range <0, 65535>.
m	Any 8-bit location for all the addressing modes allowed for the particular instruction.

SUMMARY OF FLAG OPERATION

S	Z	X	H	X	P/V	N	C

Sequence of flags in F register

OBJ CODE	SOURCE STATEMENT		OPERATION
8E	ADC	A,(HL)	Add with Carry Operand to Acc.
DD8E05	ADC	A,(IX+d)	
FD8E05	ADC	A,(IY+d)	
8F	ADC	A,A	
88	ADC	A,B	
89	ADC	A,C	
8A	ADC	A,D	
8B	ADC	A,E	
8C	ADC	A,H	
8D	ADC	A,L	
CE20	ADC	A,n	
ED4A	ADC	HL,BC	Add with Carry Reg. Pair to HL
ED5A	ADC	HL,DE	
ED6A	ADC	HL,HL	
ED7A	ADC	HL,SP	
86	ADD	A,(HL)	Add Operand to Acc.
DD8605	ADD	A,(IX+d)	
FD8605	ADD	A,(IY+d)	
87	ADD	A,A	
80	ADD	A,B	
81	ADD	A,C	
82	ADD	A,D	
83	ADD	A,E	
84	ADD	A,H	
85	ADD	A,L	
C620	ADD	A,n	
09	ADD	HL,BC	Add Reg. Pair to HL
19	ADD	HL,DE	
29	ADD	HL,HL	
39	ADD	HL,SP	
DD09	ADD	IX,BC	Add Reg. Pair to IX
DD19	ADD	IX,DE	
DD29	ADD	IX,IX	
DD39	ADD	IX,SP	
FD09	ADD	IY,BC	Add Reg. Pair to IY
FD19	ADD	IY,DE	
FD29	ADD	IY,IY	
FD39	ADD	IY,SP	
A6	AND	(HL)	Logical 'AND' of Operand and Acc.
DDA605	AND	(IX+d)	
FDA605	AND	(IY+d)	
A7	AND	A	
A0	AND	B	
A1	AND	C	
A2	AND	D	
A3	AND	E	
A4	AND	H	
A5	AND	L	
E620	AND	n	
CB46	BIT	0,(HL)	Test Bit b of Location or Reg.
DDCB0546	BIT	0,(IX+d)	
FDCB0546	BIT	0,(IY+d)	
CB47	BIT	0,A	
CB40	BIT	0,B	
CB41	BIT	0,C	
CB42	BIT	0,D	
CB43	BIT	0,E	
CB44	BIT	0,H	

Courtesy Zilog Corp.

OBJ CODE	SOURCE STATEMENT		OPERATION
CB45	BIT	0,L	Test Bit b of Location or Reg.
CB4E	BIT	1,(HL)	
DDCB054E	BIT	1,(IX+d)	
FDCB054E	BIT	1,(IY+d)	
CB4F	BIT	1,A	
CB48	BIT	1,B	
CB49	BIT	1,C	
CB4A	BIT	1,D	
CB4B	BIT	1,E	
CB4C	BIT	1,H	
CB4D	BIT	1,L	
CB56	BIT	2,(HL)	
DDCB0556	BIT	2,(IX+d)	
FDCB0556	BIT	2,(IY+d)	
CB57	BIT	2,A	
CB50	BIT	2,B	
CB51	BIT	2,C	
CB52	BIT	2,D	
CB53	BIT	2,E	
CB54	BIT	2,H	
CB55	BIT	2,L	
CB5E	BIT	3,(HL)	
DDCB055E	BIT	3,(IX+d)	
FDCB055E	BIT	3,(IY+d)	
CB5F	BIT	3,A	
CB58	BIT	3,B	
CB59	BIT	3,C	
CB5A	BIT	3,D	
CB5B	BIT	3,E	
CB5C	BIT	3,H	
CB5D	BIT	3,L	
CB66	BIT	4,(HL)	
DDCB0566	BIT	4,(IX+d)	
FDCB0566	BIT	4,(IY+d)	
CB67	BIT	4,A	
CB60	BIT	4,B	
CB61	BIT	4,C	
CB62	BIT	4,D	
CB63	BIT	4,E	
CB64	BIT	4,H	
CB65	BIT	4,L	
CB6E	BIT	5,(HL)	
DDCB056E	BIT	5,(IX+d)	
FDCB056E	BIT	5,(IY+d)	
CB6F	BIT	5,A	
CB68	BIT	5,B	
CB69	BIT	5,C	
CB6A	BIT	5,D	
CB6B	BIT	5,E	
CB6C	BIT	5,H	
CB6D	BIT	5,L	
CB76	BIT	6,(HL)	
DDCB0576	BIT	6,(IX+d)	
FDCB0576	BIT	6,(IY+d)	
CB77	BIT	6,A	
CB70	BIT	6,B	
CB71	BIT	6,C	
CB72	BIT	6,D	
CB73	BIT	6,E	
CB74	BIT	6,H	
CB75	BIT	6,L	
CB7E	BIT	7,(HL)	
DDCB057E	BIT	7,(IX+d)	
FDCB057E	BIT	7,(IY+d)	Test Bit b Location or Reg.
CB7F	BIT	7,A	
CB78	BIT	7,B	
CB79	BIT	7,C	
CB7A	BIT	7,D	
CB7B	BIT	7,E	
CB7C	BIT	7,H	
CB7D	BIT	7,L	
DC8405	CALL	C,nn	Call Subroutine at Location nn if Condition True
FC8405	CALL	M,nn	
D48405	CALL	NC,nn	
C48405	CALL	NZ,nn	
F48405	CALL	P,nn	
EC8405	CALL	PE,nn	
E48405	CALL	PO,nn	
CC8405	CALL	Z,nn	
CD8405	CALL	nn	Unconditional Call to Subroutine at nn
3F	CCF		Complement Carry Flag
BE	CP	(HL)	Compare Operand with Acc.
DDBE05	CP	(IX+d)	
FDBE05	CP	(IY+d)	
BF	CP	A	
B8	CP	B	
B9	CP	C	
BA	CP	D	
BB	CP	E	
BC	CP	H	
BD	CP	L	
FE20	CP	n	
EDA9	CPD		Compare Location (HL) and Acc. Decrement HL and BC
EDB9	CPDR		Compare Location (HL) and Acc. Decrement HL and BC, Repeat until BC = 0
EDA1	CPI		Compare Location (HL) and Acc., Increment HL and Decrement BC
EDB1	CPIR		Compare Location (HL) and Acc. Increment HL, Decrement BC, Repeat until BC = 0
2F	CPL		Complement Acc. (1's Comp)
27	DAA		Decimal Adjust Acc.
35	DEC	(HL)	Decrement Operand
DD3505	DEC	(IX+d)	
FD3505	DEC	(IY+d)	
3D	DEC	A	
05	DEC	B	
0B	DEC	BC	
0D	DEC	C	
15	DEC	D	
1B	DEC	DE	

Courtesy Zilog Corp.

OBJ CODE	SOURCE STATEMENT		OPERATION
1D	DEC	E	Decrement Operand
25	DEC	H	
2B	DEC	HL	
DD2B	DEC	IX	
FD2B	DEC	IY	
2D	DEC	L	
3B	DEC	SP	
F3	DI		Disable Interrupts
102E	DJNZ	e	Decrement B and Jump Relative if B = 0
FB	EI		Enable Interrupts
E3	EX	(SP),HL	Exchange Location and (SP)
DDE3	EX	(SP),IX	
FDE3	EX	(SP),IY	
08	EX	AF,AF'	Exchange the Contents of AF and AF'
EB	EX	DE,HL	Exchange the Contents of DE and HL
D9	EXX		Exchange the Contents of BC,DE,HL with Contents of BC',DE',HL' Respectively
76	HALT		HALT (Wait for Interrupt or Reset)
ED46	IM	0	Set Interrupt Mode
ED56	IM	1	
ED5E	IM	2	
ED78	IN	A,(C)	Load Reg. with Input from Device (C)
ED40	IN	B,(C)	
ED48	IN	C,(C)	
ED50	IN	D,(C)	
ED58	IN	E,(C)	
ED60	IN	H,(C)	
ED68	IN	L,(C)	
34	INC	(HL)	Increment Operand
DD3405	INC	(IX+d)	
FD3405	INC	(IY+d)	
3C	INC	A	
04	INC	B	
03	INC	BC	
0C	INC	C	
14	INC	D	
13	INC	DE	
1C	INC	E	
24	INC	H	
23	INC	HL	
DD23	INC	IX	
FD23	INC	IY	
2C	INC	L	
33	INC	SP	
DB20	IN	A,(n)	Load Acc. with Input from Device n
EDAA	IND		Load Location (HL) with Input from Port (C), Decrement HL and B

OBJ CODE	SOURCE STATEMENT		OPRRTION
EDBA	INDR		Load Location (HL) with Input from Port (C), Decrement HL and Decrement B, Repeat until B 0
EDA2	INI		Load Location (HL) with Input from Port (C), Increment HL and Decrement B
EDB2	INIR		Load Location (HL) with Input from Port (C), Increment HL and Decrement B, Repeat until B = 0
E9	JP	(HL)	Unconditional Jump to Location
DDE9	JP	(IX)	
C38405	JP	nn	
FDE9	JP	(IY)	
DA8405	JP	C,nn	Jump to Location if Condition True
FA8405	JP	M,nn	
D28405	JP	NC,nn	
C28405	JP	NZ,nn	
F28405	JP	P,nn	
EA8405	JP	PE,nn	
E28405	JP	PO,nn	
CA8405	JP	Z,nn	
382E	JR	C,e	Jump Relative to PC+e if Condition True
302E	JR	NC,e	
202E	JR	NZ,e	
282E	JR	Z,e	
182E	JR	e	Unconditional Jump Relative to PC+e
02	LD	(BC),A	Load Source to Destination
12	LD	(DE),A	
77	LD	(HL),A	
70	LD	(HL),B	
71	LD	(HL),C	
72	LD	(HL),D	
73	LD	(HL),E	
74	LD	(HL),H	
75	LD	(HL),L	
3620	LD	(HL),n	
DD7705	LD	(IX+d),A	
DD7005	LD	(IX+d),B	
DD7105	LD	(IX+d),C	
DD7205	LD	(IX+d),D	
DD7305	LD	(IX+d),E	
DD7405	LD	(IX+d),H	
DD7505	LD	(IX+d),L	
DD360520	LD	(IX+d),n	
FD7705	LD	(IY+d),A	
FD7005	LD	(IY+d),B	
FD7105	LD	(IY+d),C	
FD7205	LD	(IY+d),D	
FD7305	LD	(IY+d),E	
FD7405	LD	(IY+d),H	
FD7505	LD	(IY+d),L	
FD360520	LD	(IY+d),n	
328405	LD	(nn),A	
ED438405	LD	(nn),BC	

Courtesy Zilog Corp.

OBJ CODE	SOURCE STATEMENT		OPERATION
ED538405	LD	(nn),DE	Load Source to Destination
228405	LD	(nn),HL	
DD228405	LD	(nn),IX	
FD228405	LD	(nn),IY	
ED738405	LD	(nn),SP	
0A	LD	A,(BC)	
1A	LD	A,(DE)	
7E	LD	A,(HL)	
DD7E05	LD	A,(IX+d)	
FD7E05	LD	A,(IY+d)	
3A8405	LD	A,(nn)	
7F	LD	A,A	
78	LD	A,B	
79	LD	A,C	
7A	LD	A,D	
7B	LD	A,E	
7C	LD	A,H	
ED57	LD	A,I	
7D	LD	A,L	
3E20	LD	A,n	
ED5F	LD	A,R	
46	LD	B,(HL)	
DD4605	LD	B,(IX+d)	
FD4605	LD	B,(IY+d)	
47	LD	B,A	
40	LD	B,B	
41	LD	B,C	
42	LD	B,D	
43	LD	B,E	
44	LD	B,H	
45	LD	B,L	
0620	LD	B,n	
ED4B8405	LD	BC,(nn)	
018405	LD	BC,nn	
4E	LD	C,(HL)	
DD4E05	LD	C,(IX+d)	
FD4E05	LD	C,(IY+d)	
4F	LD	C,A	
48	LD	C,B	
49	LD	C,C	
4A	LD	C,D	
4B	LD	C,E	
4C	LD	C,H	
4D	LD	C,L	
0E20	LD	C,n	
56	LD	D,(HL)	
DD5605	LD	D,(IX+d)	
FD5605	LD	D,(IY+d)	
57	LD	D,A	
50	LD	D,B	
51	LD	D,C	
52	LD	D,D	
53	LD	D,E	
54	LD	D,H	
55	LD	D,L	
1620	LD	D,n	
ED5B8405	LD	DE,(nn)	
118405	LD	DE,nn	
5E	LD	E,(HL)	
DD5E05	LD	E,(IX+d)	
FD5E05	LD	E,(IY+d)	
5F	LD	E,A	
58	LD	E,B	
59	LD	E,C	

OBJ CODE	SOURCE STATEMENT		OPERATION
5A	LD	E,D	Load Source to Destination
5B	LD	E,E	
5C	LD	E,H	
5D	LD	E,L	
1E20	LD	E,n	
66	LD	H,(HL)	
DD6605	LD	H,(IX+d)	
FD6605	LD	H,(IY+d)	
67	LD	H,A	
60	LD	H,B	
61	LD	H,C	
62	LD	H,D	
63	LD	H,E	
64	LD	H,H	
65	LD	H,L	
2620	LD	H,n	
2A8405	LD	HL,(nn)	
218405	LD	HL,nn	
ED47	LD	I,A	
DD2A8405	LD	IX,(nn)	
DD218405	LD	IX,nn	
FD2A8405	LD	IY,(nn)	
FD218405	LD	IY,nn	
6E	LD	L,(HL)	
DD6E05	LD	L,(IX+d)	
FD6E05	LD	L,(IY+d)	
6F	LD	L,A	
68	LD	L,B	
69	LD	L,C	
6A	LD	L,D	
6B	LD	L,E	
6C	LD	L,H	
6D	LD	L,L	
2E20	LD	L,n	
ED4F		LD R,A	
ED7B8405	LD	SP,(nn)	
F9	LD	SP,HL	
DDF9	LD	SP,IX	
FDF9	LD	SP,IY	
318405	LD	SP,nn	
EDA8	LDD		Load Location (DE) with Location (HL), Decrement DE,HL and BC
EDB8	LDDR		Load Location (DE) with Location (HL), Repeat until BC = 0
EDA0	LDI		Load Location (DE) with Location (HL), Increment DE,HL, Decrement BC
EDB0	LDIR		Load Location (DE) with Location (HL), Increment DE,HL, Decrement BC and Repeat until BC = 0
ED44	NEG		Negate Acc. (2's Complement)
00	NOP		No Operation
B6	OR	(HL)	Logical "OR" of Operand and Acc.
DDB605	OR	(IX+d)	

Courtesy Zilog Corp.

OBJ CODE	SOURCE STATEMENT		OPERATION
FD8605	OR	(IY+d)	Logical "OR" of Operand and Acc.
B7	OR	A	
B0	OR	B	
B1	OR	C	
B2	OR	D	
B3	OR	E	
B4	OR	H	
B5	OR	L	
F620	OR	n	
ED8B	OTDR		Load Output Port (C) with Location (HL) Decrement HL and B, Repeat until B = 0
ED83	OTIR		Load Output Port (C) with Location (HL), Increment HL, Decrement B, Repeat until B = 0
ED79	OUT	(C),A	Load Output Port (C) with Reg.
ED41	OUT	(C),B	
ED49	OUT	(C),C	
ED51	OUT	(C),D	
ED59	OUT	(C),E	
ED61	OUT	(C),H	
ED69	OUT	(C),L	
D320	OUT	(n),A	Load Output Port (n) with Acc.
EDAB	OUTD		Load Output Port (C) with Location (HL), Decrement HL and B
EDA3	OUTI		Load Output Port (C) with Location (HL), Increment HL and Decrement B
F1	POP	AF	Load Destination with Top of Stack
C1	POP	BC	
D1	POP	DE	
E1	POP	HL	
DDE1	POP	IX	
FDE1	POP	IY	
F5	PUSH	AF	Load Source to Stack
C5	PUSH	BC	
D5	PUSH	DE	
E5	PUSH	HL	
DDE5	PUSH	IX	
FDE5	PUSH	IY	
CB86	RES	0,(HL)	Reset Bit b of Operand
DDCB0586	RES	0,(IX+d)	
FDCB0586	RES	0,(IY+d)	
CB87	RES	0,A	
CB80	RES	0,B	
CB81	RES	0,C	
CB82	RES	0,D	
CB83	RES	0,E	
CB84	RES	0,H	
CB85	RES	0,L	
CB8E	RES	1,(HL)	
DDCB058E	RES	1,(IX+d)	
FDCB058E	RES	1,(IY+d)	
CB8F	RES	1,A	

OBJ CODE	SOURCE STATEMENT		OPERATION
CB88	RES	1,B	Reset Bit b of Operand
CB89	RES	1,C	
CB8A	RES	1,D	
CB8B	RES	1,E	
CB8C	RES	1,H	
CB8D	RES	1,L	
CB96	RES	2,(HL)	
DDCB0596	RES	2,(IX+d)	
FDCB0596	RES	2,(IY+d)	
CB97	RES	2,A	
CB90	RES	2,B	
CB91	RES	2,C	
CB92	RES	2,D	
CB93	RES	2,E	
CB94	RES	2,H	
CB95	RES	2,L	
CB9E	RES	3,(HL)	
DDCB059E	RES	3,(IX+d)	
FDCB059E	RES	3,(IY+d)	
CB9F	RES	3,A	
CB98	RES	3,B	
CB99	RES	3,C	
CB9A	RES	3,D	
CB9B	RES	3,E	
CB9C	RES	3,H	
CB9D	RES	3,L	
CBA6	RES	4,(HL)	
DDCB05A6	RES	4,(IX+d)	
FDCB05A6	RES	4,(IY+d)	
CBA7	RES	4,A	
CBA0	RES	4,B	
CBA1	RES	4,C	
CBA2	RES	4,D	
DBA3	RES	4,E	
CBA4	RES	4,H	
CBA5	RES	4,L	
CBAE	RES	5,(HL)	
DDCB05AE	RES	5,(IX+d)	
FDCB05AE	RES	5,(IY+d)	
CBAF	RES	5,A	
CBA8	RES	5,B	
CBA9	RES	5,C	
CBAA	RES	5,D	
CBAB	RES	5,E	
CBAC	RES	5,H	
CBAD	RES	5,L	
CBB6	RES	6,(HL)	
DDCB0586	RES	6,(IX+d)	
FDCB0586	RES	6,(IY+d)	
CBB7	RES	6,A	
CBB0	RES	6,B	
CBB1	RES	6,C	
CBB2	RES	6,D	
CBB3	RES	6,E	
CBB4	RES	6,H	
CBB5	RES	6,L	
CBBE	RES	7,(HL)	
DDCB05BE	RES	7,(IX+d)	
FDCB05BE	RES	7,(IY+d)	
CBBF	RES	7,A	
CBB8	RES	7,B	
CBB9	RES	7,C	
CBBA	RES	7,D	

Courtesy Zilog Corp.

OBJ CODE	SOURCE STATEMENT		OPERATION
CBBB	RES	7,E	Reset Bit b of Operand
CBBC	RES	7,H	
CBBD	RES	7,L	
C9	RET		Return from Subroutine
D8	RET	C	Return from Subroutine if Condition True
F8	RET	M	
D0	RET	NC	
C0	RET	NZ	
F0	RET	P	
E8	RET	PE	
E0	RET	PO	
C8	RET	Z	
ED4D	RETI		Return from Interrupt
ED45	RETN		Return from Non-Maskable Interrupt
CB16	RL	(HL)	Rotate Left Through Carry
DDCB0516	RL	(IX+d)	
FDCB0516	RL	(IY+d)	
CB17	RL	A	
CB10	RL	B	
CB11	RL	C	
CB12	RL	D	
CB13	RL	E	
CB14	RL	H	
CB15	RL	L	
17	RLA		Rotate Left Acc. Through Carry
CB06	RLC	(HL)	Rotate Left Circular
DDCB0506	RLC	(IX+d)	
FDCB0506	RLC	(IY+d)	
CB07	RLC	A	
CB00	RLC	B	
CB01	RLC	C	
CB02	RLC	D	
CB03	RLC	E	
CB04	RLC	H	
CB05	RLC	L	
07	RLCA		Rotate Left Circular Acc.
ED6F	RLD		Rotate Digit Left and Right between Acc. and and Location (HL)
CB1E	RR	(HL)	Rotate Right Through Carry
DDCB051E	RR	(IX+d)	
FDCB051E	RR	(IY+d)	
CB1F	RR	A	
CB18	RR	B	
CB19	RR	C	
CB1A	RR	D	
CB1B	RR	E	
CB1C	RR	H	
CB1D	RR	L	
1F	RRA		Rotate Right Acc. Through Carry
CB0E	RRC	(HL)	Rotate Right Circular
DDCB050E	RRC	(IX+d)	
FDCB050E	RRC	(IY+d)	
CB0F	RRC	A	

OBJ CODE	SOURCE STATEMENT		OPERATION
CB08	RRC	B	Rotate Right Circular
CB09	RRC	C	
CB0A	RRC	D	
CB0B	RRC	E	
CB0C	RRC	H	
CB0D	RRC	L	
0F	RRCA		Rotate Right Circular Acc.
ED67	RRD		Rotate Digit Right and Left Between Acc. and Location (HL)
C7	RST	00H	Restart to Location
CF	RST	08H	
D7	RST	10H	
DF	RST	18H	
E7	RST	20H	
EF	RST	28H	
F7	RST	30H	
FF	RST	38H	
9E	SBC	A,(HL)	Subtract Operand from Acc. with Carry
DD9E05	SBC	A,(IX+d)	
FD9E05	SBC	A,(IY+d)	
9F	SBC	A,A	
98	SBC	A,B	
99	SBC	A,C	
9A	SBC	A,D	
9B	SBC	A,E	
9C	SBC	A,H	
9D	SBC	A,L	
DE20	SBC	A,n	
ED42	SBC	HL,BC	
ED52	SBC	HL,DE	
ED62	SBC	HL,HL	
ED72	SBC	HL,SP	
37	SCF		Set Carry Flag (C = 1)
CBC6	SET	0,(HL)	Set Bit b of Location
DDCB05C6	SET	0,(IX+d)	
FDCB05C6	SET	0,(IY+d)	
CBC7	SET	0,A	
CBC0	SET	0,B	
CBC1	SET	0,C	
CBC2	SET	0,D	
CBC3	SET	0,E	
CBC4	SET	0,H	
CBC5	SET	0,L	
CBCE	SET	1,(HL)	
DDCB05CE	SET	1,(IX+d)	
FDCB05CE	SET	1,(IY+d)	
CBCF	SET	1,A	
CBC8	SET	1,B	
CBC9	SET	1,C	
CBCA	SET	1,D	
CBCB	SET	1,E	
CBCC	SET	1,H	
CBCD	SET	1,L	
CBD6	SET	2,(HL)	
DDCB05D6	SET	2,(IX+d)	
FDCB05D6	SET	2,(IY+d)	
CBD7	SET	2,A	
CBD0	SET	2,B	
CBD1	SET	2,C	
CBD2	SET	2,D	

Courtesy Zilog Corp.

OBJ CODE	SOURCE STATEMENT		OPERATION
CBD3	SET	2,E	Set Bit b of Location
CBD4	SET	2,H	
CBD5	SET	2,L	
CBD8	SET	3,B	
CBDE	SET	3,(HL)	
DDCB05DE	SET	3,(IX+d)	
FDCB05DE	SET	3,(IY+d)	
CBDF	SET	3,A	
CBD9	SET	3,C	
CBDA	SET	3,D	
CBDB	SET	3,E	
CBDC	SET	3,H	
CBDD	SET	3,L	
CBE6	SET	4,(HL)	
DDCB05E6	SET	4,(IX+d)	
FDCB05E6	SET	4,(IY+d)	
CBE7	SET	4,A	
CBE0	SET	4,B	
CBE1	SET	4,C	
CBE2	SET	4,D	
CBE3	SET	4,E	
CBE4	SET	4,H	
CBE5	SET	4,L	
CBEE	SET	5,(HL)	
DDCB05EE	SET	5,(IX+d)	
FDCB05EE	SET	5,(IY+d)	
CBEF	SET	5,A	
CBE8	SET	5,B	
CBE9	SET	5,C	
CBEA	SET	5,D	
CBEB	SET	5,E	
CBEC	SET	5,H	
CBED	SET	5,L	
CBF6	SET	6,(HL)	
DDCB05F6	SET	6,(IX+d)	
FDCB05F6	SET	6,(IY+d)	
CBF7	SET	6,A	
CBF0	SET	6,B	
CBF1	SET	6,C	
CBF2	SET	6,D	
CBF3	SET	6,E	
CBF4	SET	6,H	
CBF5	SET	6,L	
CBFE	SET	7,(HL)	
DDCB05FE	SET	7,(IX+d)	
FDCB05FE	SET	7,(IY+d)	
CBFF	SET	7,A	
CBF8	SET	7,B	
CBF9	SET	7,C	
CBFA	SET	7,D	
CBFB	SET	7,E	
CBFC	SET	7,H	
CBFD	SET	7,L	
CB26	SLA	(HL)	Shift Operand Left Arithmetic
DDCB0526	SLA	(IX+d)	
FDCB0526	SLA	(IY+d)	
CB27	SLA	A	
CB20	SLA	B	
CB21	SLA	C	
CB22	SLA	D	
CB23	SLA	E	
CB24	SLA	H	
CB25	SLA	L	

OBJ CODE	SOURCE STATEMENT		OPERATION
CB2E	SRA	(HL)	Shift Operand Right Arithmetic
DDCB052E	SRA	(IX+d)	
FDCB052E	SRA	(IY+d)	
CB2F	SRA	A	
CB28	SRA	B	
CB29	SRA	C	
CB2A	SRA	D	
CB2B	SRA	E	
CB2C	SRA	H	
CB2D	SRA	L	
CB3E	SRL	(HL)	Shift Operand Right Logical
DDCB053E	SRL	(IX+d)	
FDCB053E	SRL	(IY+d)	
CB3F	SRL	A	
CB38	SRL	B	
CB39	SRL	C	
CB3A	SRL	D	
CB3B	SRL	E	
CB3C	SRL	H	
CB3D	SRL	L	
96	SUB	(HL)	Subtract Operand from Acc.
DD9605	SUB	(IX+d)	
FD9605	SUB	(IY+d)	
97	SUB	A	
90	SUB	B	
91	SUB	C	
92	SUB	D	
93	SUB	E	
94	SUB	H	
95	SUB	L	
D620	SUB	n	
AE	XOR	(HL)	Exclusive "OR" Operand and Acc.
DDAE05	XOR	(IX+d)	
FDAE05	XOR	(IY+d)	
AF	XOR	A	
A8	XOR	B	
A9	XOR	C	
AA	XOR	D	
AB	XOR	E	
AC	XOR	H	
AD	XOR	L	
EE20	XOR	n	

Courtesy Zilog Corp.

SECTION ZX

Description of the Z-80 Instruction Set

Load Instructions	ZX-1
Exchange Instructions, Block Transfer and Searches	ZX-5
Arithmetic Instructions	ZX-7
Logic Instructions	ZX-8
Increment and Decrement	ZX-9
Rotate Instructions	ZX-12
Bit Set and Test	ZX-15
Jump Instructions	ZX-16
I/O Instructions	ZX-20

H

Z-80 INSTRUCTION SET

The Z-80 microprocessor was designed to be software compatible with the Intel 8080 microprocessor. In fact, the 8080 instruction set is a Subset of the instructions for the Z-80 although the Intel mnemonics are not the same. In the study of the Z-80 instruction set we will examine those instructions which are identical to the 8080 instructions as well as those which are unique to the Z-80.

Z-80 8 Bit Load Instruction

A. One Byte Load Instructions

Z-80		8080	
LD r,r'	r ← r'	MOV REG_1,REG_2	(REG_1) ← (REG_2)
LD r,(HL)	r ← (HL)	MOV REG,M	(REG_2) ← [(HL)]
LD (HL),r	(HL) ← r	MOV M,REG	[(HL)] ← [REG]
LD A,(BC)	A ← (BC)	LDAX B	(A) ← [(BC)]
LD A,(DE)	A ← (DE)	LDAX D	(A) ← [(BC)]
LD (BC),A	(BC) ← A	STAX B	[(BC)] ← [A]
LD (DE),A	(DE) ← A	STAX D	[(DE)] ← [A]

B. Two Byte Load Instructions

LD r,n	r ← n	MVI REG, DATA	(REG) ← DATA
LD (HL), n	(HL) ← n	MVI M,DATA	[(HL)] ← DATA

C. Three Byte Load Instructions

LD A,(nn)	A ← (nn)	LDA (addr)	(A) ← (addr)
LD (nn),A	(nn) ← A	STA (addr)	(addr) ← (A)

D. Unique Z-80 Three Byte Load Instructions

LD r,(IX+d) r ← (IX+d)

LD r,(IY+d) r ← (IY+d)

The contents of the memory location specified by the current contents of the index register (IX) or (IY) plus displacement (d) designated by the third byte of the instruction are loaded into the selected register (r).

LD (IX+d),r (IX+d) ← r

LD (IY+d),r (IY+d) ← r

The contents of register (r) are loaded into the memory location specified by index register (IX) or (IY) plus displacement (d).

E. Unique Z-80 Four Byte Load Instructions

LD (IX+d),n (IX+d) ← n

LD (IY+d),n (IY+d) ← n

The 8 bit operand n (immediate data) is loaded into the address location specified by index register (IX) or (IY) plus a displacement (d).

F. Unique Two Byte Z-80 Load Instructions

LD A,I A ← I

LD A,R A ← R

Contents of Interrupt register (I) or Refresh register (R) are loaded into the accumulator.

LD I,A I ← A

LD R,A R ← A

Contents of accumulator are loaded into the I or the R register.

Z-80 16 Bit Load Instructions

A. Four Byte 16 Bit Load Instructions

LD IX,nn IX ← nn

LD IY,nn IY ← nn

16 Bits of data (nn) are loaded into IX or IY (Byte 3 and Byte 4 identify immediate data)

LD IX (nn) IX_H ← (nn+1) IX_L ← (nn)

LD IY (nn) IY_H ← (nn+1) IY_L ← (nn)

16 Bits of data are loaded from two consecutive memory locations into Index Register (IX) or (IY). (Byte 3 and Byte 4 identify Adr_L and Adr_H respectively.)

LD (nn),dd (nn+1) ← dd_H

Load into address locations nn+1 and nn the high and low order bytes respectively of register pair dd.

LD dd,(nn) dd_H ← (nn+1) dd_L ← (nn)

Load into register pair dd the contents of address locations nn and nn+1 respectively.

B. Three Byte 16 Bit Load Instructions

Z-80		8080	
LD dd,nn	dd ← nn	LXI RP,Data 16	(RP) ← Data 16
LD HL,(nn)	H ← (nn+1) L ← (nn)	LHLD	(H) ← (addr+1) (L) ← (addr)
LD (nn),HL	(nn+1) ← H (nn) ← L	SHLD	(addr+1) ← H (addr) ← L

C. One Byte 16 Bit Load Instructions

Z-80		8080	
LDSP,HL	SP ← HL	SPHL	(SP) ← (HL)
PUSH qq	(SP-2) ← qq_L (SP-1) ← qq_H	PUSH RP	[(SP-2)] ← RP_H [(SP-1)] ← RP_L
POP qq	qq_L ← (SP) qq_H ← (SP+1)	POP RP	RP_L ← [(SP)] RP_H ← [(SP+1)]

D. Z-80 Sixteen Bit Load Instructions

LD SP,IX SP ← IX

LD SP,IX SP ← IY

Contents of IX, IY are loaded into Stack Pointer.

PUSH IX (SP-2) ← IX_L (SP-1) ← IX_H

PUSH IY (SP-2) ← IY_L (SP-1) ← IY_H

The Stack Pointer initially at SP is decremented automatically prior to loading each byte of data.

POP IX IX_H ← (SP) IX_L ← (SP-1)

POP IY IY_H ← (SP) IY_L ← (SP-1)

The Stack Pointer, initially at SP is incremented automatically after reading each byte of data.

Z-80 Exchange Instructions

Only two of the six Z-80 exchange instructions have a counterpart in the 8080 instruction set. These are:

Z80	8080	Z80	8080
EX DE, HL	XCHG	EX(SP),HL	XTHL

EX DE,HL DE ↔ HL

EX AF,AF' AF ↔ AF'

EX (SP),HL H ↔ (SP+1), L ↔ (SP)

EX (SP),IX IX_H ↔ (SP+1), IX_L ↔ (SP)

EX (SP),IY IY_H ↔ (SP+1), IY_L ↔ (SP)

EXX BC ↔ BC' DE ↔ DE' HL ↔ HL'

In each of these instructions the contents of specified registers are exchanged with a single instruction.

Block Transfer and Search Instructions

LDI (DE) ← (HL) DE ← DE+1,HL ← HL+1, BC ← BC-1

A byte of data is transferred from the memory location specified by the HL register pair to the memory location specified by the DE pair. HL and DE are incremented, BC (Byte Counter) is decremented.

LDIR (DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1

Data is transferred from the memory location specified by HL to the location specified by DE. HL and DE are incremented. If BC is not zero the program counter, PC, is decremented by two and the instruction is repeated.

LDD (DE) ← (HL) DE ← DE-1, HL ← HL-1, BC ← BC-1

Data is transferred from the memory location specified by HL to the location specified by DE. Both HL and DE are decremented as well as BC.

LDDR (DE) ← (HL) DE ← DE-1, HL ← HL-1, BC ← BC-1

Data is tranferred from the memory location specified by HL to the location specified by DE. All three register pairs are decremented. If BC is not zero, the program counter is decremented by two, and the instruction is repeated.

CPI A-(HL), HL ← HL + 1, BC ← BC - 1

The contents of the memory location addressed by HL is compared to the contents of the Accumulator. If the comparison is true, a condition bit is set for the condition which is true.

CPIR A-(HL), HL ← HL + 1, BC ← BC - 1

The contents of memory location addressed by HL is compared to the accumulator as in previous instruction. HL is incremented and the Byte Counter (BC) is decremented. If decrementing causes BC to go to zero, or if A = (HL), the instruction is terminated, otherwise the program counter is decremented by two and the instruction is repeated.

CPD A - (HL), HL ← HL - 1, BC ← BC - 1

The contents of memory location addressed by HL is compared to the contents of the Accumulator. A condition bit is set for each condition of true compare. HL and BC are decremented.

CPDR A - (HL), HL ← HL - 1, BC ← BC - 1

(HL) is compared with the contents of the accumulator. As before, a condition bit is set for each true compare condition. HL and BC are decremented. If BC = 0, or, A = (HL), the instruction is terminated, otherwise, the program counter is decremented by two causing the instruction to be repeated.

8 Bit Arithmetic Instructions

A. Add Operations

Z-80		8080	
ADD A,r	A ← A+r	ADD REG	A ← (A) + (REG)
ADD A,n	A ← A+n	ADI	A ← (A) + (DATA)
ADD A,(HL)	A ← A+(HL)	ADD M	A ← (A) + (M)

The contents of a specified register, a memory location specified by HL or immediate data described by the second byte of the two byte add instructions are added to the contents of the accumulator.

ADD A,(IX+D)	A ← A + (IX+D)
ADD A,(IY+D)	A ← A + (IY+D)

The contents of the memory addressed by (IX+D) or (IY+D) are added to the contents of the accumulator.

B. Add with Carry Operations

Z-80

ADC A,r	A ← A+r+CY	ADC	A ← (A) + (REG) + (CY)
ADC A,n	A ← A+n+CY	ACI	A ← (A) + DATA + (CY)
ADC A,(HL)	A ← A+(HL)+CY	ADC M	A ← (A) + (M) + (CY)

The contents of a specified register, memory location, or immediate data plus the carry is added to the accumulator.

ADC A,(IX+D)	A ← A + (IX+D)+ Cy
ADC A,(IY+D)	A ← A + (IY+D)+ Cy

The contents of the memory location addressed by (IX+D) or (IY+D) plus the carry are added to the accumulator.

C. Subtraction Operations

SUB r	A ← A-r	SUB REG	A ← (A) - (REG)
SUB n	A ← A-n	SUI	A ← (A) - DATA
SUB (HL)	A ← A-(HL)	SUB M	A ← (A) - (M)

These instructions are parallel to the ADD operations.

```
SUB(IX+D)   A ← A-(IX+D)
SUB(IY+D)   A ← A-(IY+D)
```

These instructions parallel the Index Register ADD operations.

```
SBC A,(IX+D) A ← (IX+D)+CY
SBC A,(IY+D) A ← (IY+D)+CY
```

These instructions parallel the Index Register ADC operations.

```
SBC A,r     A ← A-r-CY      SBB REG   A ← (A)-(REG)-(CY)
SBC A,n     A ← A-n-CY      SBI       A ← (A)-DATA-(CY)
SBC A,(HL)  A ← A-(HL)-CY   SBB M     A ← (A)-(M)-(CY)
```

These instructions parallel the ADC instructions.

8 BIT LOGIC INSTRUCTIONS

A. Basic Logic Instructions

```
AND r     A ← A∧r     ANA REG     (A) ← (A) · (REG)
AND n     A ← A∧n     ANI DATA    (A) ← (A) · DATA
AND (HL)  A ← A∧(HL)  ANA M       (A) ← (A) · (M)
```

Contents of the Accumulator are logically "ANDed" with the contents of a specified register, memory location or with immediate data.

```
OR r      A ← AVr     ORA REG     (A) ← (A) + (REG)
OR n      A ← AVn     ORI DATA    (A) ← (A) + DATA
OR (HL)   A ← AV(HL)  ORA M       (A) ← (A) + (M)
```

Contents of the Accumulator are logically "ORed" with the contents of a specified register, memory location or with immediate data.

```
XOR r     A ← A⊕r     XRA REG     (A) ← (A) ⊕ (REG)
XOR n     A ← A⊕n     XRI DATA    (A) ← (A) ⊕ DATA
XOR (HL)  A ← A⊕(HL)  XRA M       (A) ← (A) ⊕ (M)
```

The contents of the Accumulator are logically "Exclusive ORed" with the contents of a specified register, memory location or with immediate data.

CP r	A - r	CMP REG	(A) - (REG)
CP n	A - n	CPI DATA	(A) - DATA
CP (HL)	A - (HL)	CMP M	(A) - (M)

The contents of the specified register, memory location or immediate data is subtracted from the contents of the Accumulator setting or resetting condition flags as determined by the results of the subtraction.

B. Index Register Logic Operations

AND (IX+D)	A ← A∧(IX+D)
AND (IY+D)	A ← A∧(IY+D)
OR (IX+D)	A ← AV(IX+D)
OR (IY+D)	A ← AV(IY+D)

The content of the Accumulator are "ANDed" or "ORed" with the contents of the memory location addressed by (IX+D) or (IY+D).

XOR (IX+D)	A ← A ⊕ (IX+D)
XOR (IY+D)	A ← A ⊕ (IY+D)

A logical "Exclusive OR" is executed between the memory location addressed by (IX+D) or (IY+D) and the Accumulator.

CP (IX+D)	A - (IX+D)
CP (IY+D)	A - (IY+D)

The contents of the memory location addressed by (IX+D) or (IY+D) is compare with the contents of the Accumulator. Condition flags are set or reset depending on the results of the comparison.

C. Increment and Decrement Instructions

Z-80		8080	
INC r	r ← r+1	INR REG	(REG) ← (REG) + 1
INC (HL)	(HL) ← (HL)+1	INR M	(M) ← (M) + 1
DEC r	r ← r-1	DCR REG	(REG) ← (REG) - 1
DEC (HL)	(HL) ← (HL)-1	DCR M	(M) ← (M) - 1

The content of a specified register r, or the memory location addressed by HL are either incremented or decremented by one.

D. Index Register Increment and Decrement Instructions

INC (IX+D) (IX+D) ← (IX+D) + 1
INC (IY+D) (IY+D) ← (IY+D) + 1

DEC (IX+D) (IX+D) ← (IX+D) - 1
DEC (IY+D) (IY+D) ← (IY+D) - 1

The memory location addressed by either (IX+D) or (IY+D) is incremented, or decremented by one.

Arithmetic and CPU Control Instructions

DAA —

This instruction conditionally adjusts the Accumulator for BCD addition and subraction operations.

CPL A ← $\bar{A}$

Contents of Accumulator are complemented (1's compl.).

NEG A ← 0 - A

Contents of the Accumulator are subtracted from zero (2's complement of A).

CCF CY ← $\overline{CY}$

Carry flag in flag register is inverted.

SCF CY ← 1

Carry flag is set to a one.

NOP —

CPU performs no operation during machine cycle.

HALT

The HALT instruction suspends CPU operations until a subsequent interrupt or reset is received. The processor executes NOP's during a HALT in order to maintain refresh logic.

DI IFF ← 0

DI disables the maskable interrupt by resetting interrupt enable flip-flops IFF 1 and IFF 2.

EI IFF ← 1

EI enables the maskable interrupt by setting the interrupt enable flip-flops IFF 1 and IFF 2.

IM 0 ——

The IM 0 instruction sets interrupt mode 0. In this mode the interrupting device can insert any instruction onto the data bus for execution by the CPU.

IM 2 ——

The interrupt instruction IM 2 sets interrupt mode 2. This mode permits an indirect call to any location in memory. The upper eight bits are the contents of the Interrupt Vector Register, I. The lower eight bits are supplied by the interrupting device.

Sixteen Bit Arithmetic Operations

ADD HL, ss HL ← HL + ss DAD RP

The content of a selected register pair SS are added to the contents of the HL register pair without carry.

ADC HL, ss HL ← HL + ss + CY

The contents of SS plus the carry from the flag register are added to the contents of HL.

SBC HL, ss HL ← HL - ss - CY

The contents of register pair SS and the carry CY are subtracted from the contents of the HL reg pair.

ADD IX, pp IX ← IX + pp

The contents of any register pair pp, (BC, DE, IX, or SP) are added to the contents of index register IX.

ADD IY, rr IY ← IY + rr

The contents of any register pair rr, (BC, DE, IY or SP) are added to index register IY.

INC ss ss ← ss + 1

The contents of any register pair SS (BC, DE, HL, SP) are incremented by 1.

INC IX IX ← IX + 1
INC IY IX ← IY + 1

The contents of index register IX or IY are incremented.

DEC ss ss ← ss - 1

The contents of any register pair ss, (BC, DE, HL, SP) are decremented by 1.

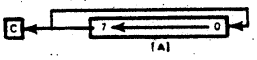
DEC IX IX ← IX - 1
DEC IY IY ← IY - 1

The contents of Index Register IX or IY are decremented.

Rotate Instructions

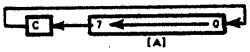
A. Accumulator Rotate Instructions

Z-80 | 8080

RLCA 

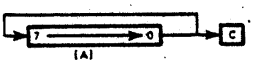
RLC

The contents of the accumulator are rotated left one position. Contents of b7 are copied into carry flag and position b_0.

RLA 

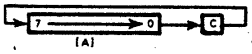
RAL

The contents of the accumulator are rotated left one position, b_7 is copied into the carry flag and the carry is copied into position b_0.

RRCA 

RRC

The contents of the accumulator are rotated to the right one position, b_0 is copied into the carry flag and the carry is copied into position b_7.

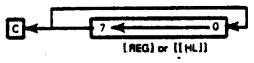
RRA 

RAR

The contents of the accumulator are rotated to the right. b_0 is copied into the carry flag. The carry is copied into bit position b_7.

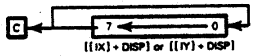
B. Register and Memory Rotate Instructions

RLC r



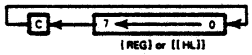
Contents of any 8 bit register r are rotated left on position. The contents of bit position b7 are copied into the carry flag and the carry is copied into bit position b_0.

RLC (HL)
RLC (IX+D)
RLC (IY+D)



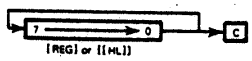
The contents of the memory location addressed by (HL), (IX+D), or (IY+D) is rotated left. b_7 is copied into carry flag and the carry is copies into position b_0.

RL r
RL (HL)
RL (IX+D)
RL (IY+D)



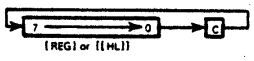
The contents of the register specified by r, or the memory location addressed by HL, IX+D or IY+D are rotated left one position, through the carry flag.

RRC r
RRC (HL)
RRC (IX+D)
RRC (IY+D)



The contents of a register specified by r, or the memory location addressed by HL, IX+D or IY+D are rotated right one position. The contents of b_0 are copied into the carry flag and the carry is copied into b_7.

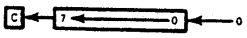
RR r
RR (HL)
RR (IX+D)
RR (IY+D)



The contents of a register specified by r, or the memory location addressed by HL, IX+D or IY+D are rotated right one position through the carry flag.

Z-80 Shift Instructions

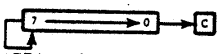
SLA r
SLA (HL)
SLA (IX+D)
SLA (IY+D)



Arithmetic Shift

Contents of the register r, or the memory location addressed by HL, IX+D, or IY+D are shifted left one position. A zero is copied into position b_0.

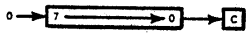
SRA r
SRA (HL)
SRA (IX+D)
SRA (IY+D)



Arithmetic Shift

Contents of register r, or the memory location addressed by HL, IX+D, or IY+D are shifted right one position. Contents of bit position 7 are unchanged.

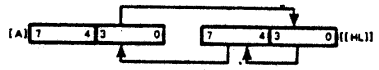
SRL r
SRL (HL)
SRL (IX+D)
SRL (IY+D)



Logic Shift

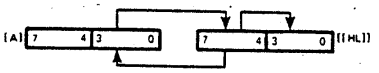
Contents of register r, or the memory location addressed by HL, IX+D, or IY+D are shift right one position. A zero is entered into bit position 7.

RLD



Contents of the low order 4 bits (3-0) of the memory location addressed by (HL) are copied into high order 4 bits (7-4). The high order 4 bits (7-4) are copied into the low order 4 bits of the accumulator (3-0), these are, in turn, copied into the low order 4 bits (3-0) of the memory location.

RRD



The contents of the low order 4 bits (3-0) of the memory location addressed by (HL) are copied into the lower order bits (3-0) of the accumulator. The low order bits (3-0) of the accumulator are rotated into the high order bits (7-0) of the memory location and these, in turn, are copied into the high order bits (7-4) of the memory location.

Bit Set & Test Instructions

A. Test Instructions

Bit b, r	$Z \leftarrow \bar{r}_b$
Bit b, (HL)	$Z \leftarrow (\overline{HL})_b$
Bit b, (IX+D)	$Z \leftarrow (\overline{IX+D})_b$
Bit b, (IY+D)	$Z \leftarrow (\overline{IY+D})_b$

After execution of the instructions, the Z flag will contain the complement of the indicated bit within the register specified or the memory location pointed to by HL, IX+D or IY+D. The indicated bit is not affected.

B. Bit Set, Reset Instructions

SET b, r	$r_b \leftarrow 1$
SET b, (HL)	$(HL)_b \leftarrow 1$
SET b, (IX+D)	$(IX+D)_b \leftarrow 1$
SET b, (IY+D)	$(IY+D)_b \leftarrow 1$

After completion of the instruction the designated bit in the register or memory location addressed will be set.

RES b, r	$r_b \leftarrow 0$
RES b, (HL)	$(HL)_b \leftarrow 0$
RES b, (IX+D)	$(IX+D)_b \leftarrow 0$
RES b, (IY+D)	$(IY+D)_b \leftarrow 0$

These instructions are analoqous to the bit set instructions Set b, m.

Jump Instructions

A. Unconditional Jump - Direct

Z-80		8080	
JP nn	PC ← nn	JMP (Address)	(PC) ← (Address)

The address specified by the 2nd and 3rd bytes of the instruction is loaded into the program counter PC and a jump to that address is initiated.

B. Conditional Jump - Direct

Z-80		8080	
JP NZ, nn	If CC True PC ← nn	JNZ	(addr)
JP Z, nn		JZ	(addr)
JP NC, nn		JNC	(addr)
JP C, nn		JC	(addr)
JP PO, nn		JPO	(addr)
JP PE, nn		JPE	(addr)
JP P, nn		JP	(addr)
JP M, nn		JM	(addr)

If condition is true, the address specified by bytes 2 and 3 of the instruction are loaded into the program counter - PC, otherwise the program counter is incremented as usual.

C. Relative Jump

JR e PC ← PC+e

This instruction permits relative branching to other segments of a program specified by a displacement e. The displacement is limited to a range $-126 \leq e \leq +129$. The next instruction is fetched from the location designated by the new contents of the PC.

D. Conditional Relative Jump

JR C, e	PC ← PC+e
JR NC, e	
JR Z, e	
JR NZ, e	

If the specified condition is true, the next instruction is fetched from the location identified by the new contents of the PC.

E. Indirect Jump

Z-80		8080	
JP (HL)	PC ← HL	PCHL	PC ← (HL)

The PC is loaded with the contents of the HL register pair. The next instruction is fetched from the location identified by the new contents of the PC.

JP (IX)	PC ← IX
JP (IY)	PC ← IY

The PC is loaded with the contents of the IX (IY) register pair. The next instruction is fetched from the location designated by the new contents of the PC.

F. Relative Condition Jump

DJNZ, e PC ← PC+e If B ≠ 0

This instruction permits relative branching to a location designated by the displacement e, relative to the PC, if the B register is not zero. On execution of the instruction, the B register is decremented and the displacement e is added to the PC. If the condition is true. In this case the next instruction will be fetched from the location designated by the new contents of the PC, otherwise the next instruction to be executed will be the one following this instruction.

CALL AND RETURN INSTRUCTIONS

A. Unconditional Call

Z-80		8080	
CALL nn	(SP-1) ← PC_H	CALL addr	[(SP-1)] ← PC_H
	(SP-2) ← PC_L		[(SP-2)] ← PC_L
	SP ← SP-2		
	PC ← nn		

On execution, the current contents of PC are pushed onto the top of the external stack memory. The address of the called routine is loaded into the PC. The next instruction is fetched from the called location.

B. Conditional Call

CALL NZ, nn	(SP-1) ← PC_H	CALL NZ, addr	[(SP-1)] ← PC_H
CALL Z, nn	(SP-2) ← PC_L	CALL Z, addr	[(SP-2)] ← PC_L
CALL NC, nn	PC ← nn	CALL NC, addr	
CALL C, nn	SP ← SP-2	CALL C, addr	
CALL PO, nn		CALL PO, addr	
CALL PE, nn		CALL PE, addr	
CALL P, nn		CALL P, addr	
CALL M, nn		CALL M, addr	

This instruction executes a call to the designated address if the specified condition is true.

C. Unconditional Return

RET	PC_L ← (SP)	RET	PC_L ← [(SP)]
	PC_H ← (SP+1)		PC_H ← [(SP+1)]

Control is returned to the program at a point desigrated by external stock memory. The content of the location specified by the stack pointer is loaded to PC_L, the stack pointer is incremented and the next higher address location is loaded to PC_H.

D. Conditional Return

Z-80		8080	
RET NZ	$PC_L \leftarrow (SP)$	RET NZ	$PC_L \leftarrow [(SP)]$
RET Z	$PC_H \leftarrow (SP+1)$	RET Z	$PC_H \leftarrow [(SP+1)]$
RET NC		RET NC	
RET C		RET C	
RET PO		RET PO	
RET PE		RET PE	
RET P		RET P	
RET M		RET M	

Execution of this instruction causes a return to the program of a point designated by the external stack memory if the specified condition is true.

E. Return from Interrupt

RETI

The return from interrupt instruction is identical to an unconditional return except it also controls the interrupt enable out control signal, used for priority interrupt structures.

F. Return from Non Maskable Interrupt

RETN

This instruction is used at the end of a service routine for non maskable interrupts and executes an unconditional return.

G. Restart Instruction

RST p RST n

This is a special one byte call to one of eight locations in page zero (lowest 256 bytes of memory) specified by the table below.

p (n)	±	±
00H	000	11 - - - 111
08H	001	11 - - - 111
10H	010	11 - - - 111
18H	011	11 - - - 111
20H	100	11 - - - 111
28H	101	11 - - - 111
30H	110	11 - - - 111
38H	111	11 - - - 111

INPUT AND OUTPUT INSTRUCTIONS

A. Input to the Accumulator

IN A (n) A ← (n) IN port A ← [(port addr)]

One byte of data from the selected port (n) is written to the accumulator.

B. Input to a Selected Register

IN r, (C) r ← (C)

The contents of the selected port designated by the C register are input to one of the six working registers (B, C, D, E, H, L,) or the accumulator. The selected register is specified by the second byte of the instruction (01 ← r → 000).

C. Input to Memory and Increment

INI (HL) ← (C) B ← B-1, HL ← HL+1

This instruction inputs a byte of data from a port specified by the C register writes it to a memory location designated by the HL register. The HL register then is incremented. The B register operates as a byte counter and is autoatically decremented. When the B register equals zero, the Z flag is set.

D. Input to Memory Increment and repeat

INIR (HL) ← (C) B ← B-1, HL ← HL+1

This instruction is identical to the INI instruction except, if the B register is not zero, the PC is decremented by two and the instruction is repeated. If decrementing causes the B register to go to zero the instruction is terminated and the next instruction is executed.

E. Input to Memory and Decrement

IND (HL) ← (C) B ← B-1 HL ← HL-1

This instruction is identical to the INI instruction except, the AL register is decremented instead of being incremented.

F. Input to Memory Decrement and Repeat

INDR (HL) ← (C) B ← B-1 HL ← HL-1

This instruction is identical to the IND instruction except, the instruction is repeated until the byte counter B equals zero.

G. Output from the Accumulator

OUT (n), A (n) ← A OUT (port addr)

One byte of data is written from the Accumulator to a port selected by the port address (n).

H. Output from a Selected Register

OUT (C), r (C) ← r

One byte of data is written out to a port selected by the C register from a register designated by r. The register r may be the accumulator or one of the six working registers (B,C,D, E,H,L).

I. Output from Memory and Increment

OUTI (c) ← HL, B ← B-1 HL ← HL+1

This instruction moves one byte of data to the CPU from a memory location designated by the HL register pair and writes it to a port selected by the C register. The HL register pair is then incremented and the B register, used as a byte counter, is decremented.

J. Output from Memory Increment and Repeat

OTIR (C) ← (HL) B ← B-1, HL ← HL-1

This instruction is identical to the OUTI except, the instruction is repeated until the B register equals zero; at this point, the instruction is terminated.

K. Output from Memory and Decrement

OUTD (C) ← HL, B ← B-1, HL ← HL-1

This instruction is identical to the OUTI instruction except, the HL register pair is decremented instead of being incremented.

L. Output from Memory Decrement and Repeat

OTDR (C) ← (HL) B ← B-1 HL ← HL-1

This instruction is identical to the OUTD instruction except, the instruction is repeated until the B register equals zero. At this point the instruction is terminated.

SECTION CTC

The Counter Timer Circuit CTC-1

CTC Timing CTC-4

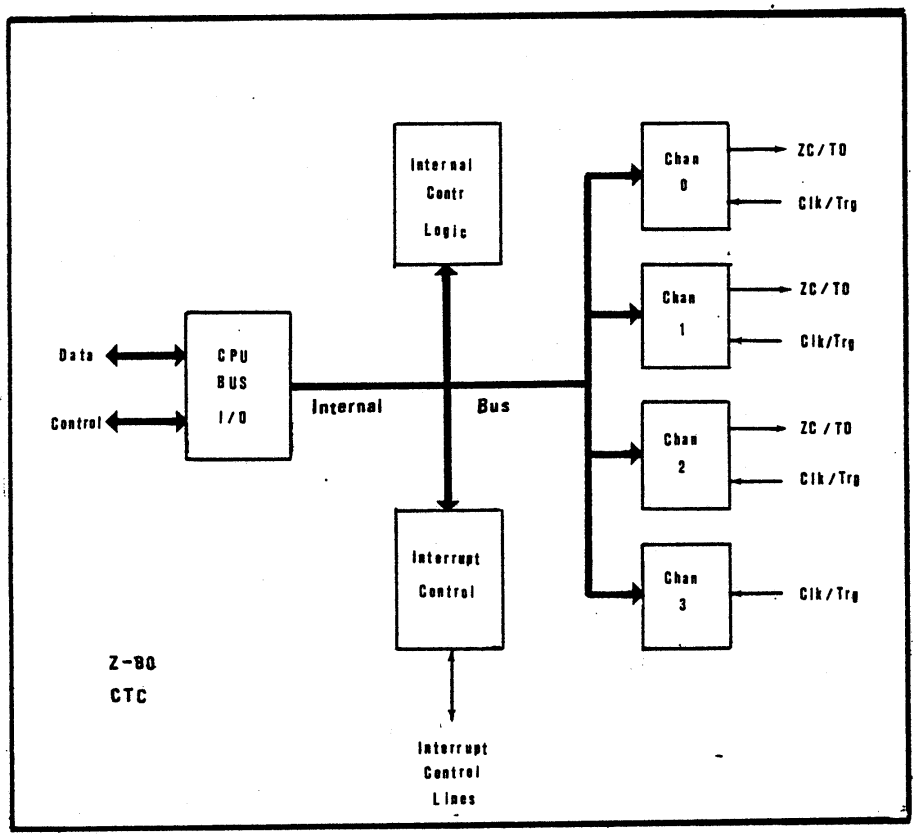
CTC Operating Modes CTC-7

CTC Interface CTC-9

Programming The CTC CTC-10

Z-80 CTC - COUNTER TIMER CIRCUIT

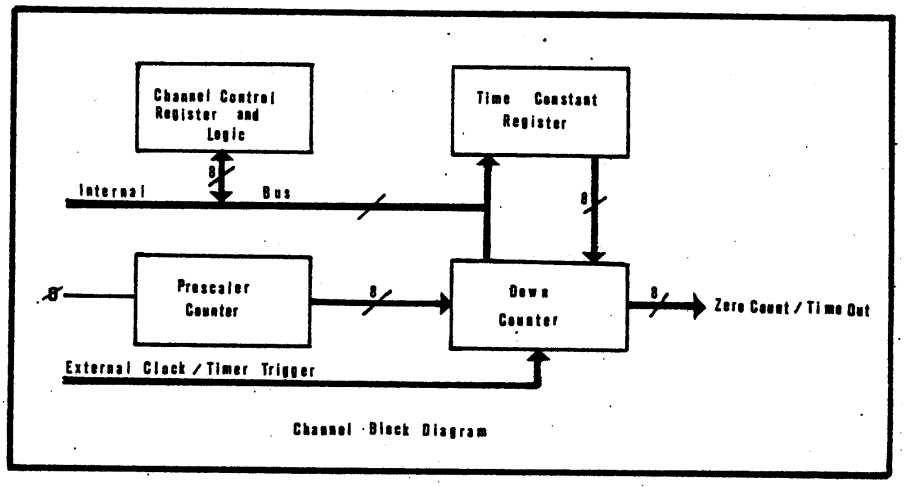
This circuit is a programmable four channel device which provides counting and timing functions. The four independent channels of the device can be configured for operation in various modes as required.



Counter Timer Circuit - CTC

CTC - Channel Logic

The circuit for each channel consists of two registers, two counters and control logic as shown:



Channel Block Diagram

a/ Channel Control Register - Selects mode and conditions for channel operation based on 8-bit word loaded to register by CPU.

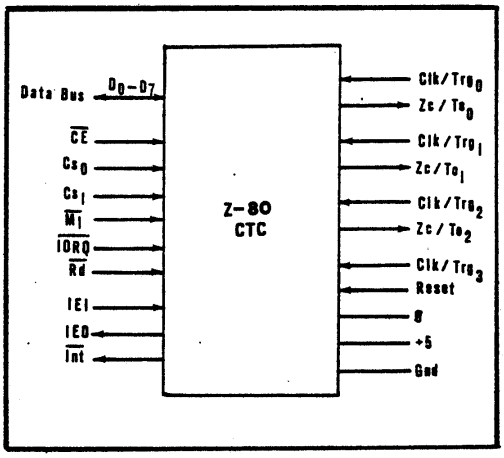
b/ Time Constant Register - An 8-bit word loaded by the CPU into this register initializes the Down Counter and reloads at a count of zero.

c/ Down Counter - This 8-bit counter counter counts down from its initialized value until the count is zero, at which time it reloads and repeats the count.

d/ Prescaler - An 8-bit counter that divides the system clock by 16 or 256 (used in timer mode only).

Z-80 CTC

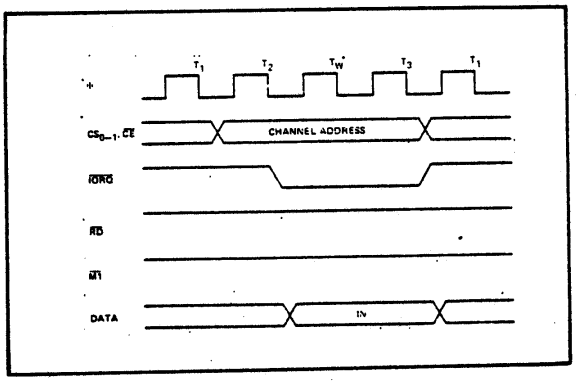
CTC Pin Description



$CLK/Trig_{0-3}$	-	Ext Clock or Timer Strobe Input
ZC/To_{0-2}	-	Channel - Zero Count or Time Out Output
C_{S1} - C_{S2}	-	Channel Select Input
D_7 - D_0	-	CPU Data Bus
$\overline{CE}$	-	Chip Enable
ø	-	System Clock
$\overline{M1}$	-	M1 Cycle from CPU
$\overline{IORQ}$	-	Input/Output Request from CPU
$\overline{RD}$	-	Read Mode Control from CPU
IEI	-	Interrupt Enable Out
IEO	-	Interrupt Enable Out
$\overline{INT}$	-	Interrupt Request to CPU
$\overline{Reset}$	-	RESET - stops all channels from Counting. (Resets channel Interrupt Enable bits on all channels)

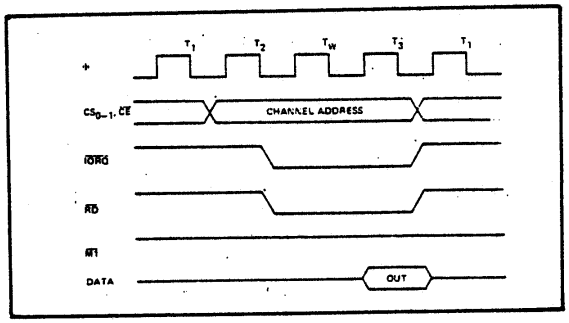
CTC Timing

CTC Write Cycle



The Channel Control Word, the Time Constant, and the Interrupt Vector are each loaded during a CTC Write Cycle. Except for an automatically inserted (TW*), no other wait states are allowed for writing to the CTC. A write signal is internally generated if $\overline{RD}$ is inactive.

CTC Read Cycle

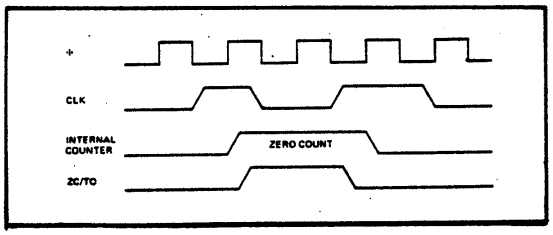


The timing shown here is for reading the Down Counter in the Counter Mode. The value placed on the data bus identifies the number of external positive pulses occuring prior to the rising edge of T_2.

External Clock or Time Out Activation

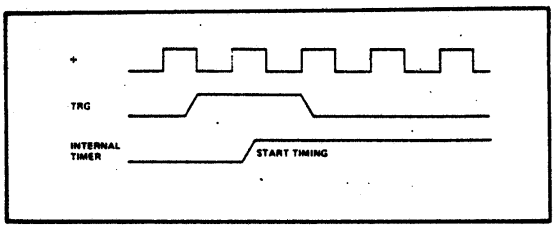
Counter Mode

In this mode, the rising edge of the clock input causes the counter to be decremented. Since the counter is synchronous with Ø, the set up time required prior to the activation of Ø must be met. (approx. 150 ns)



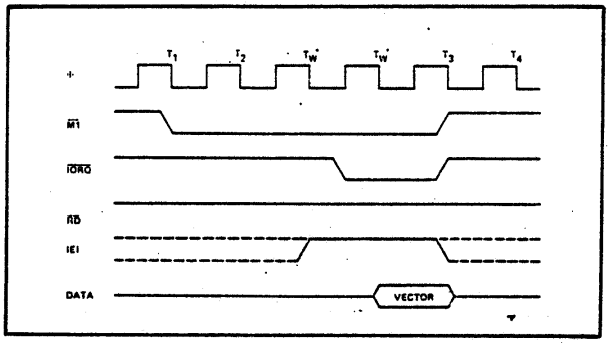
Timer Mode

In the Time Mode, the prescaler will be enabled by the rising (or falling) edge on the TRG inputs depending on the slope selected. When timing is to start, with respect to the next rising edge of Ø, set up time requirements must be met. (210 ns Min)



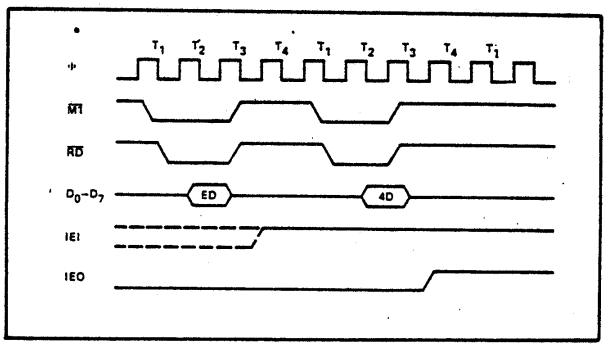
CTC Interrupt Acknowledge Cycle

Interrupt acknowledge ($\overline{IORQ}$ and $\overline{M1}$) only occurs during an M1 cycle, when $\overline{M1}$ is active. When $\overline{IORQ}$ becomes active, the highest priority device having its (IEI) input active, will put the contents of its Interrupt Vector Register onto the Data Bus.



CTC Return from Interrupt

The software routine for servicing the CTC will be terminated by the two byte RTI instruction. The CTC will decode the first byte of the op code "ED" causing IEI to become an active high on the channel being serviced at the time. If the following byte of the instruction is "4D" (RETI OPCODE = ED4D), the channel being serviced will be re-initialized and its IEO will become an active high.



CTC Operating Mode Selection

The mode of operation of the CTC is determined by the characteristics of the Control Word stored in the Channel Control Register. Control is specified at the bit level as shown:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Int En	Mode	Rnge	Slpe	Trig	LD Time Cnst	Rst	1

D_7 - Channel Interrupt Enable (D_7=1). Enable occurs each time Down Counter reaches zero count.

D_6 - Timer Mode - D_6=0 Down Counter is clocked by prescaler. Period of counter is $\Delta = t_c \cdot P \cdot TC$ t_c=system clock period. P=Prescale of 16 or 256. TC=8bit programmable time constant.

Counter Mode - D_6=1 Down Counter is clocked by external clock signal (Prescaler is not used)

D_5 - Range - Prescale = 16 for D_5 = Ø

Prescale = 256 for D_5 = 1

D_4 - Slope - for D_4=0 The negative edge of the external clock/trigger decrements the counter, or starts the timer. D_4=1 The positive edge of clock/trigger decrements the counter or starts the timer.

D_3 - Trigger Valid (Timer Mode Only) - D_3=0 Timer starts operation on rising edge of the machine cycle. T_2 clock pulse following the one that loads TC to Time Constant Register. D_3=1 The external trigger is valid for starting timer operation after the rising edge of the T_2 clock pulse following the one that loads the time constant

D_2 - for D_2=0 No time constant will be loaded into the Time Constant Register following the Channel Control Word for D_2=1 The Time Constant for the Down Counter will be the next word written to the selected channel. Time constants are loaded only after the current count has been completed.

D_1 - D_1=0 Channel continues counting. D_1=1 If D_2=1 channel will resume operation after loading a time constant. Otherwise, a new control word must be loaded.

D_0 - D_0=1 Control Word to be stored in Control Register.

Loading a Time Constant

An 8-bit time constant is loaded into the Time Constant Register immediately following a Control Word in which D_2=1. (All zeros are a time constant of 256)

Loading the Channel Interrupt Vector

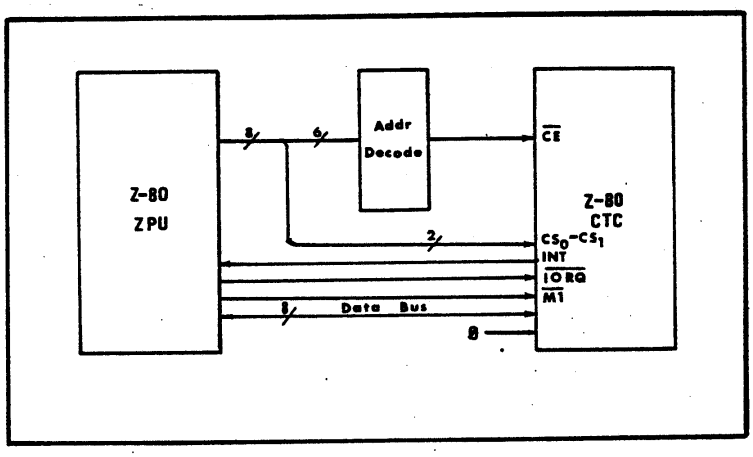
The Z-80 CPU requires an 8-bit Interrupt Vector be supplied by the interrupting channel to form the proper address for the interrupt service routine. Format of the Interrupt Vector is illustrated:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	X	X	0

1. X = These bits are inserted by the highest priority channel requesting an interrupt.

2. (D_0 is always zero in an Interrupt Vector.)

CTC Interface



The interface shown permits the selection of any one of the four channels using A_0 and A_1 as channel select inputs. Decoding for the channel ports could be as shown in the Channel Select Table.

Channel Select Decode									
Chip Select-$\overline{CS}$						C_{S1}	C_{S2}		
A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Port	Chan
0	0	0	0	1	0	0	0	08H	0
0	0	0	0	1	0	0	1	09H	1
0	0	0	0	1	0	1	0	0AH	2
0	0	0	0	1	0	1	1	0BH	3

Programming the CTC

As an example, the CTC will be programmed to count the number of events occuring in a specified interval of time at a selected peripheral device. Channel 0 of the CTC will be programmed as a count down counter to count the number of events and Channel 3 will be programmed to time out the required time interval. Termination of the time out initiates an interrupt service routine which calculates the number of events that have occurred.

```
CTC     LD A, 55H       Form Counter Control Word
        OUT 08H, A      Output Counter Control Word
        LD A, 00H       Initialize Counter, for
        OUT 08H, A      maximum count of 256
        LD A, A7        Form Timer Control Word
        OUT 0BH, A      Output to Timer Control Register
        LD A, 00H       Set Timer for maximum
        OUT 0BH, A      time interval
        LD A, 38H       Send Interrupt Vector
        OUT 0BH, A      to Timer Vector Register
```

Question: Based on the control words sent to the Counter and Timer Channels, what was the mode of operation of each? If Page 0 was selected for the interrupt Vector, to what location in Page 0 was the Interrupt Vectored?

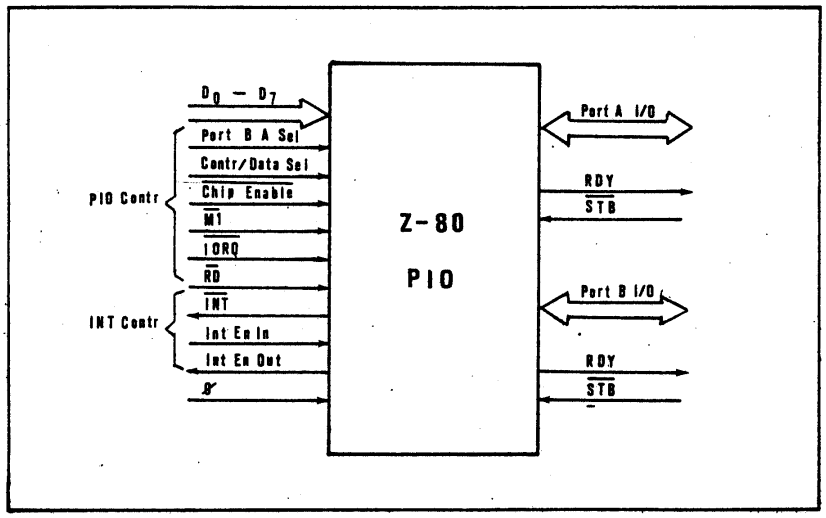
SECTION PIO

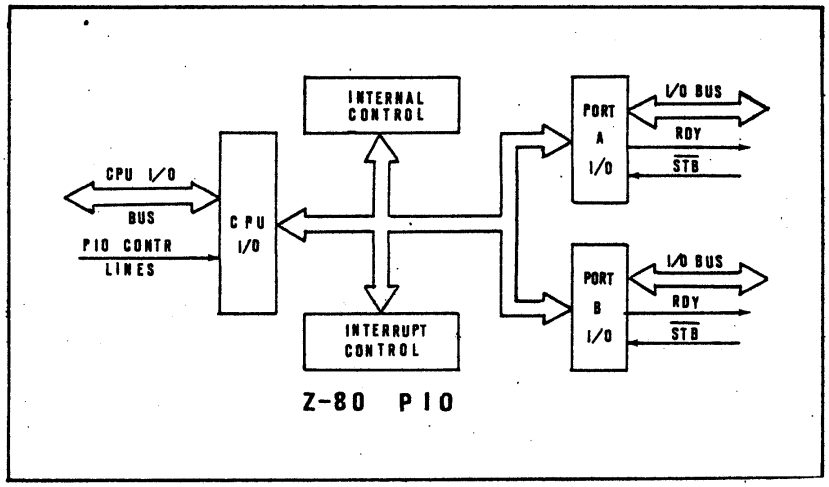
Z-80 PIO	PIO-1
Modes of Operation	PIO-3
Programming The PIO	PIO-5
I/O Port Interface	PIO-6

J

Z-80 PERIPHERAL INPUT/OUTPUT CONTROLLER - PIO

PIO Block Diagrams





PIO Port Logic

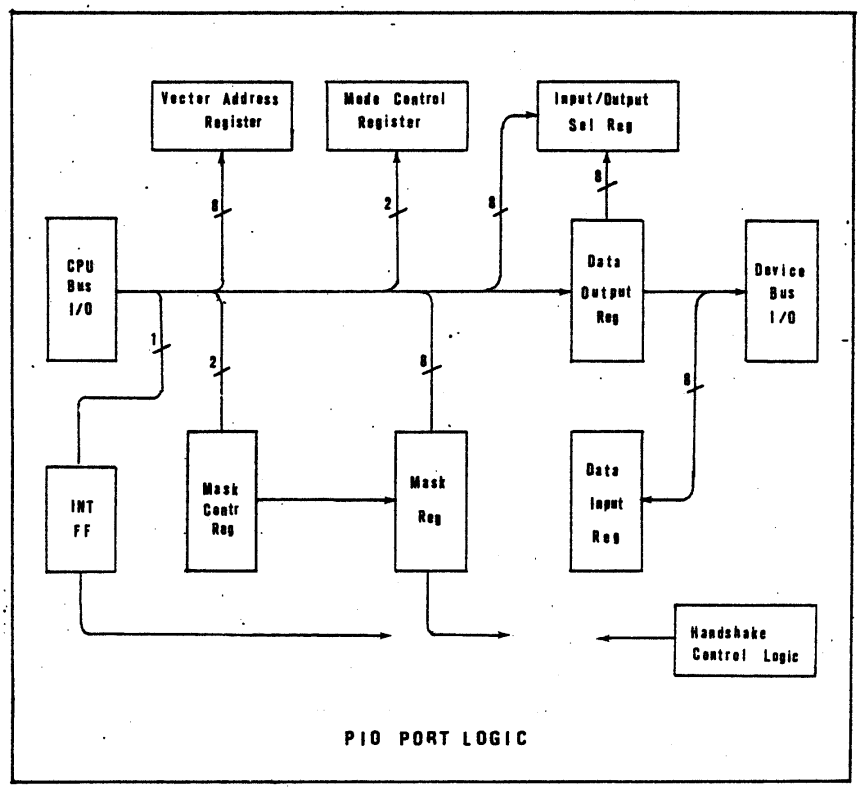
The Z-80 PIO port logic contains and internal data/logic control bus and six registers per port, with "handshake" control logic.

a. 8 Bit Data Output Register
b. 8 Bit Data Input Register
c. 8 Bit I/O Select Register
d. 8 Bit Mask Register
e. 2 Bit Mask Control Register
f. 2 Bit Mode Control Register

Additional registers and flip flops in each port include:

a. 8 Bit Port Vector Address Register
b. Port Interrupt Flip Flop

A block diagram of the port logic is shown below.



PIO PORT LOGIC

PIO Modes of Operation

The desired mode of operation is established by writing a control word to the PIO in the following format.

M1	M0	X	X	1	1	1	1

Mode Selection Table

D7	D6	Mode
0	0	0 - Output
0	1	1 - Input
1	0	2 - Bidirectional
1	1	3 - Control

Mode 0 - In the Output Mode, the output cycle is initiated by an output instruction from the CPU. The CPU $\overline{WR}$ signal latches data into the PIO Output Register. Termination of $\overline{WR}$ activates the RDY handshake signal which remains high untill the peripheral device signal $\overline{STB}$ is received. The rising edge of $\overline{STB}$ generates an interrupt to the CPU and deactivates the RDY signal. A very simple output port timing structure can be effected by tying the RDY output to the $\overline{STB}$ input. The common signal can be used to latch data to the output device and initiate an interrupt as well.

Mode 1 - In the Input Mode, the input cycle is started by an $\overline{STB}$ from the interrupting device. The transition of this signal to a low loads data from the peripheral into the Data Input Register. The rising edge of the strobe initiates the interrupt, and disables the RDY signal indicating the Input Data Register is full. During the interrupt subroutine the CPU read signal will cause the RDY to go high, indicating the Data Input Register is ready to accept new data.

Mode 2 - Mode 2 is a bi-directional mode which uses all four handshake signals. Only Port A can be used for the bi-directional mode. Port A handshake signals are for output control, and Port B handshake signals are for input control. Due to the bi-directional nature of the mode, data from the Port A Output Data Register is allowed on the port data bus only when $\overline{STB}$ is active.

Mode 3 - Operation in this mode is intended for status and control applications and does not use the handshake signals. When Mode 3 is selected, the next control word defines which of the port data bus lines will be inputs and which will be outputs. The format is shown:

I/O_7	I/O_6	I/O_5	I/O_4	I/O_3	I/O_2	I/O_1	I/O_0

Valid only in Mode 3

Data may be written to or read from a port in this mode. When reading a port the data returned to the CPU will consist of those bits assigned as inputs, plus those bits assigned as outputs.

Programming The PIO

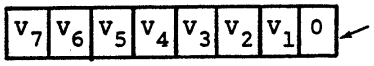
The Z-80 PIO resets automatically when power is applied. The reset conditions are as follows:

1. Both port Mask Registers are reset.
2. Port Data Bus lines enter the high impedance state and the RDY signals become inactive (low).
3. Vector Address Registers are not reset.
4. Both Interrupt Flip-Flops are reset.
5. Both port Output Registers are reset.

The PIO can also be reset by applying the M1 signal without $\overline{RD}$ or $\overline{IORQ}$ being present. The PIO will remain reset until it receives a control word from the CPU.

Forming The Interrupt Vector

Normally the PIO operates in Interrupt Mode 2, which requires an address vector to be supplied by the interrupting device. The desired interrupt vector is entered into the PIO by writing a control word to the selected port PIO.

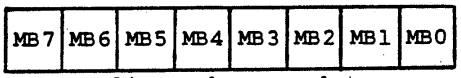


The interrupt vector is automatically loaded into the Vector Address Register.

Interrupt Control Word Format

D7 = 1 Sets Interrupt Enable F.F.

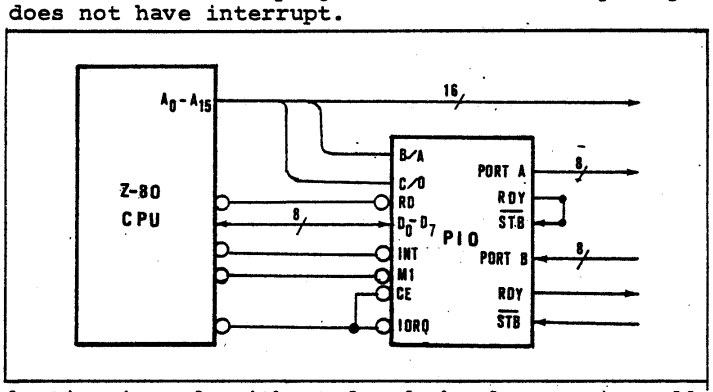
D6, D5, and D4 are used in Mode 3 only. D6 defines the logic operation for masking interrupts. D5 defines the active polarity of the port data bus lines to be monitored, and if D4 = 1, a mask will immediately follow the interrupt control word. It will be formated as shown:



Only those port lines whose mask bits are Zero will be monitored for generating an interrupt.

I/O Port Interface and Selection

The minimium system I/O shown below is being used to illustrate the techniques used to interface a PIO to the CPU. In this example, Port A is configured as an Output Port and Port B as an Input Port. Port B is programmed with interrupt capabilities. Port A does not have interrupt.



Port selection is made without decode hardware using address bits A_o and A_1, as follows:

Port Address	A_1A_o	Port Selection
00	00	Output Port A
01	01	Input Port B
02	10	Output Control Port A
03	11	Input Control Port B

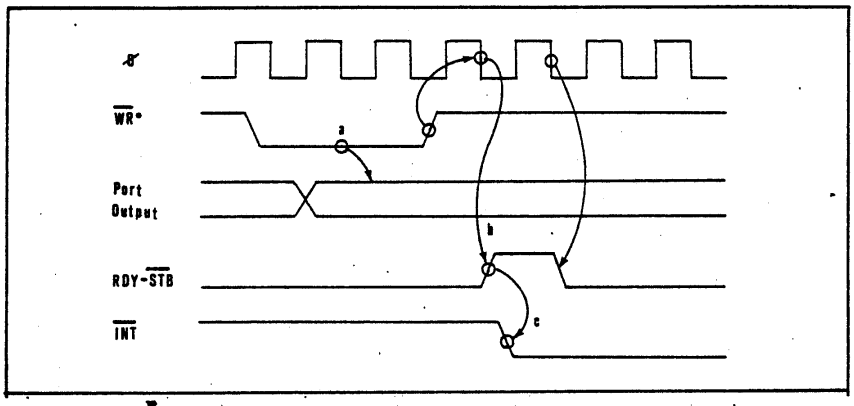
Initialization of the PIO can be programmed as follows:

```
LD A, ØF        Set Output Mode for Port A
Out Ø2,A        Load I/O Select Reg. Port A
LD A, CF        Set Input Mode for Port B
Out Ø3,A        Load I/O Select Reg. Port B
LD A, 87        Form Interrupt Control Word, Port B
Out Ø3,A        Enable Port B Interrupt
```

Timing

Output Timing - Port A

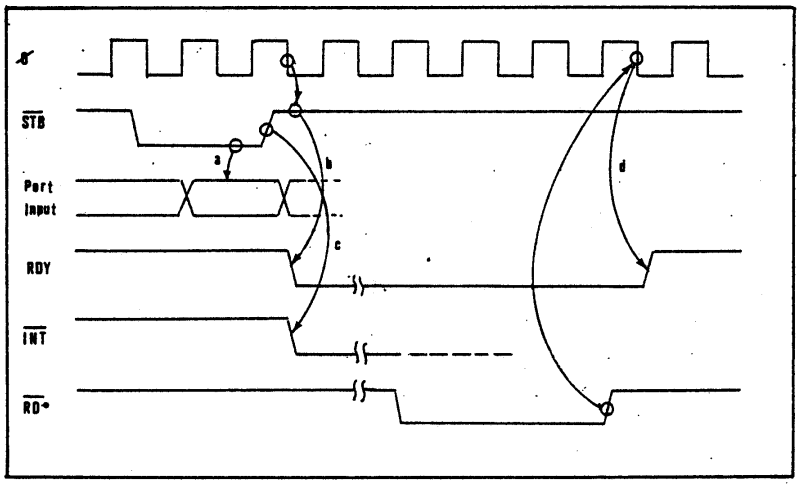
The timing and handshake control illustrated eliminates the need for a strobe from the output device. RDY is connected directly to the $\overline{STB}$ input to provide the necessary control.



$\overline{WR}* = RD \cdot \overline{CE} \cdot \overline{C/D} \cdot \overline{IORQ}$

a - Data to Port A Output Register.
b - Ready signal to device. "Data ready for transfer"
c - Interrupt enabled. (if INT FF set.)

Input Timing - Port B

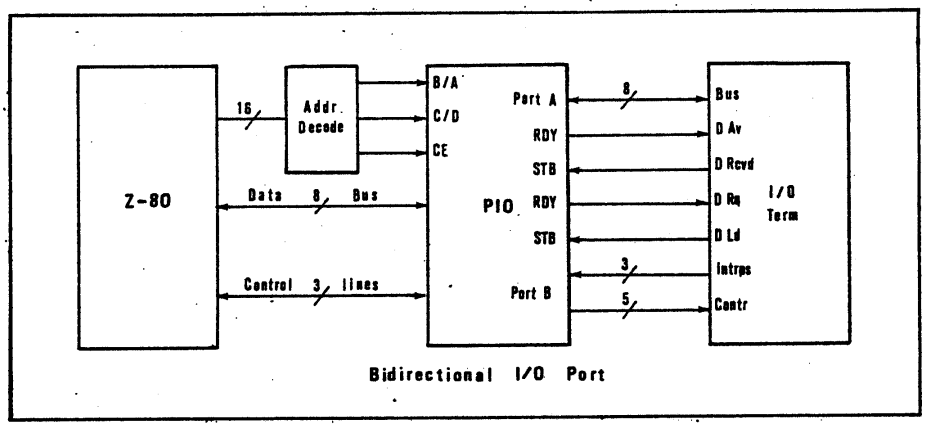


$\overline{RD}* = \overline{RD} \cdot \overline{CE} \cdot \overline{C/D} \cdot \overline{IORQ}$

a - Data from device into Data Input Register
b - Ready signal low. "Input Register Full;"
c - Interrupt enabled.
d - Ready signal high. "Input Register Empty."

I/O Interface - Bi-directional Port

In this application, the Z-80 PIO is programmed to operate in Mode 2 (bi-directional mode). Data transfer is via the Port A Data Bus. Port A handshake signals are used for output control. Port B handshake signals are used for input control. Port B must be set for bit control (Mode 3).



Bidirectional I/O Port

Timing for the Bi-directional mode is almost identical to that prescribed for the input and output modes, except that data is allowed out onto the A Data Bus only when the A strobe is low. The rising edge of this strobe can be used to latch the data into the peripheral device. Since the peripheral must not gate data onto the bus while A $\overline{STB}$ is active, the B $\overline{STB}$ is used to latch this data.

Initializing the PIO

Both ports of the PIO would have to be initialized for operation in the bi-directional mode; Port A as the Data Port and Port B as the Control Port. Assume the following port assignments have been made:

C0 = Port A Data
C1 = Port B Data
C2 = Port A Control
C3 = Port B Control

The decoder for this assignment could decode as follows:

$\overline{CE}$ = Decode $A_7 - A_2$

C/D = A_1

B/A = A_0

Port A would require the following Control Words:

Mode Select

7	6	5	4	3	2	1	0	
1	0	X	X	1	1	1	1	Mode 2

Interrupt Vector Select

7	6	5	4	3	2	1	0	
V_7	V_6	V_5	V_4	V_3	V_2	V_1	V_0	Designated Vector

Interrupt Control Word

7	6	5	4	3	2	1	0	
1	X	X	X	0	1	1	1	Set Interrupt F.F.

Port B would be initialized in the bit control mode with these Control Words:

Mode Select

7	6	5	4	3	2	1	0	
1	1	X	X	1	1	1	1	Mode 3

Pin Direction Mask

7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	1

Interrupt Inputs - D7,D3,D0
Device Control Outputs - D6,D5,D4,D2,D1

Interrupt Vector

7	6	5	4	3	2	1	0
V_7	V_6	V_5	V_4	V_3	V_2	V_1	V_0

Designated Vector

Interrupt Control Word

7	6	5	4	3	2	1	0
1	0	1	1	0	1	1	1

Control Word set for active high individual interrupts.

Interrupt Mask

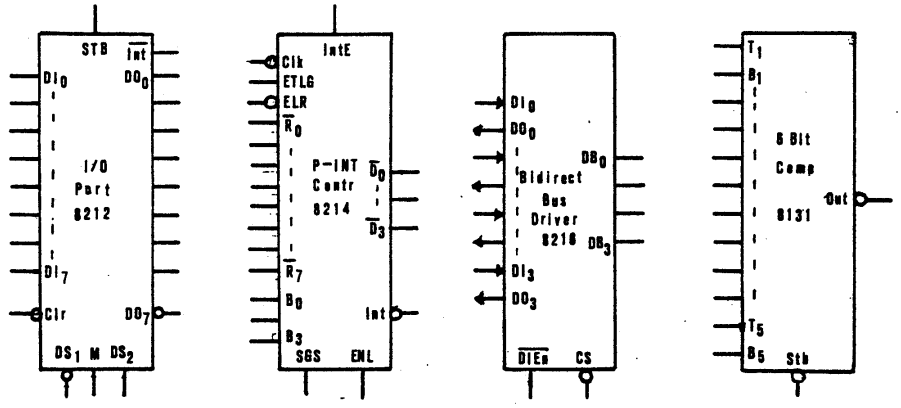
7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	0

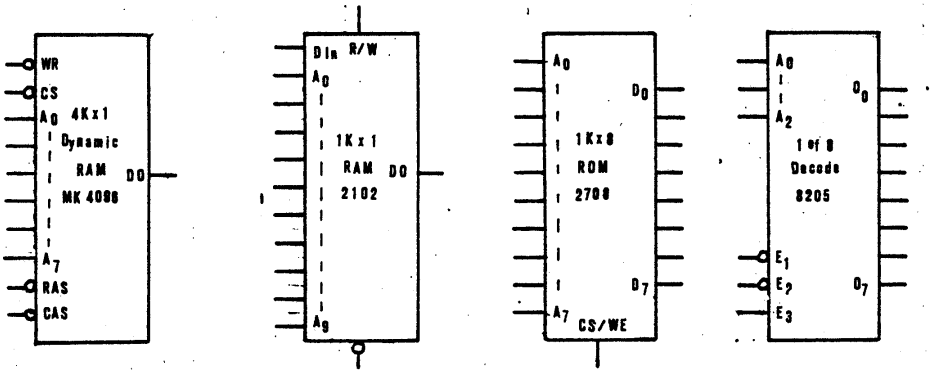
SECTION CKT

Typical Microprocessor Interface Chips CKT-1

K

TYPICAL MICROPROCESSOR INTERFACE AND MEMORY CHIPS





SECTION MA

Memory Applications	MA-1
RAM and ROM Interface Circuit	MA-2
Interfacing Dynamic ROMS	MA-3
Interface Timing	MA-6
Slow Memory Interface	MA-7

Memory Applications

Most microprocessors use both Read Write memories (RAM) and Read Only Memories (ROM). Data that is expected to be changed frequently should be stored in RAM. Data that is not likely to be changed, such as programs, look-up tables, monitors and other similar data, would be written into ROM.

Processors with modest memory requirements can use static RAMS for temporary storage. Static RAMs are easily interfaced and do not require memory refresh. Where large memories are needed, dynamic RAMs are more effective. They require less physical space, consume less power, and are often faster than the static RAMs. Interface requirements for these memories will be discussed.

Static RAM and ROM Interface

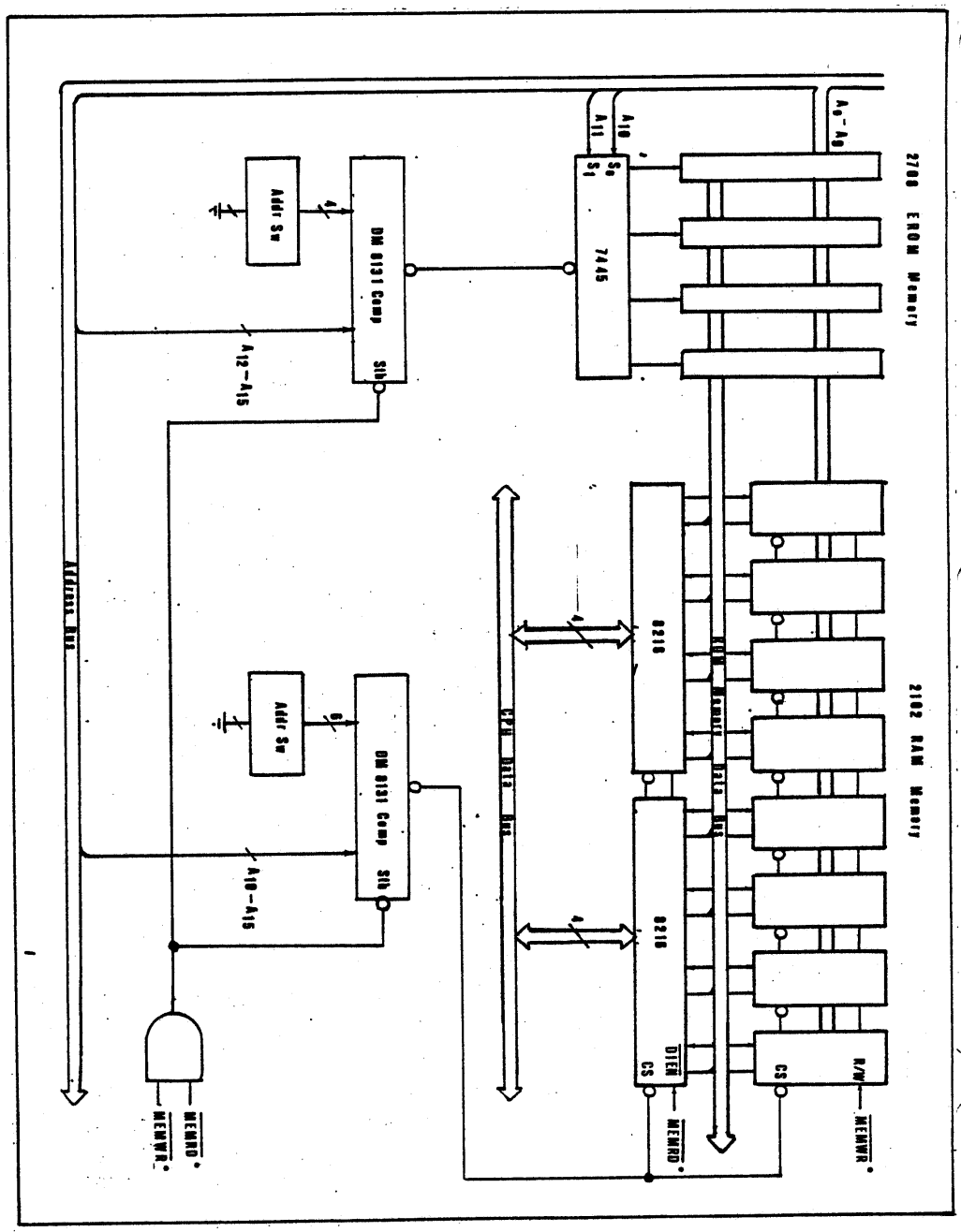
Interface for these types of memory are straightforward and will be illustrated by example.

Example:

memory system consisting of 1K of static RAM and 4K of EROM will be interfaced with the Z-80 CPU.

Requirements:

4 - 2708 1K x 8 bit EROMS
8 - 2102 1K x 1 bit static RAMs
2 - 8216 Bidirectional 4 Bit Bus Drivers
2 - DM8131 6 Bit Address Comparators
1 - 7455 Dual 4 to 1 Decoder


2708 EROM Memory
2102 RAM Memory
Addr Sw
DM 8131 Comp
7445
8216
Address Bus
CPU Data Bus
RAM Memory Data Bus
MEMRD*
MEMWR*
DIEN
CS
R/W

Interfacing Dynamic RAMs to the Z-80 System

The primary requirement for interfacing dynamic RAM to microprocessor systems is some means for refreshing the data stored in the RAM at repeated intervals no greater than 2 milliseconds in duration. This requirement makes the interface design more complex for dynamic RAMs than it is for static RAMs.

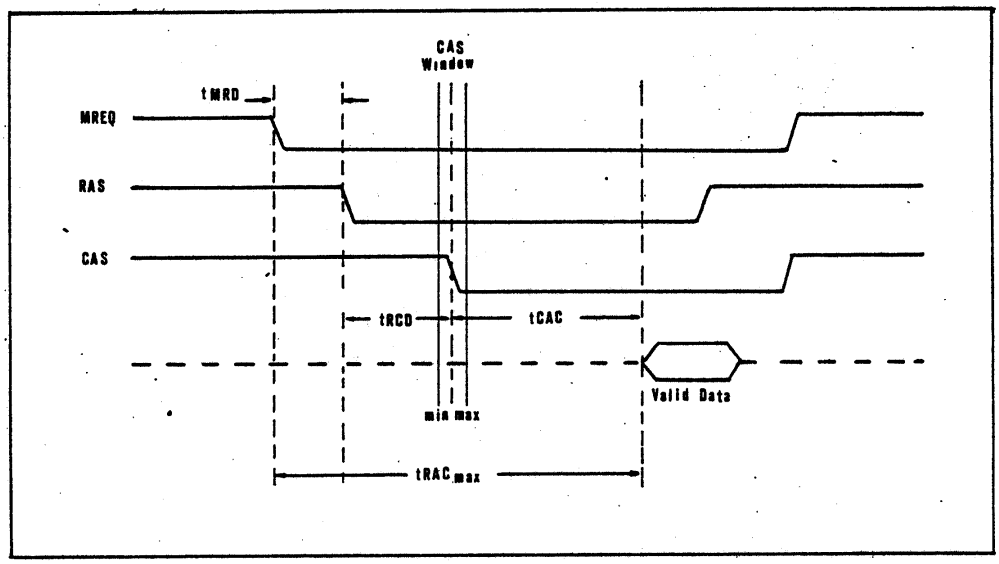
Dynamic RAM interface with the Z-80 has been simplified considerably by providing for automatic refresh during the instruction OP Code Fetch cycle. During T_3 and T_4 of this cycle a dedicated line, $\overline{RFSH}$ is activated to initiate the memory refresh operation. At this time a 7 bit Refresh Vector register is automatically incremented during the memory cycle and points to the next row to be refreshed when another Op Code Fetch cycle occurs. Since there are only 64 rows in a 4K RAM and 128 rows in a 16K RAM, refresh of the entire memory will be accomplished in less than 2 msec. This entire process is completely transparent to the user.

Dynamic RAM Addressing

To select unique bit locations within a 4K RAM chip requires 12 address lines. A 16K RAM chip would need 14 address lines. To accomodate this number of lines in a 16 pin package, it is necessary to divide the address lines into two equal groups, Row Addresses and Column Addresses. Each address group is applied to the input lines in sequence, Row Address first followed by the Column Address; this is accomplished with a switching Address Multiplexer. The address information is then latched into the RAM by applying two clock strobes in succession. The Row Address Strobe ($\overline{RAS}$) latches the Row Address information and the Column Address Strobe ($\overline{CAS}$) latches the Column Address information. When $\overline{RAS}$ activates one of the 64 rows in a 4K RAM, all of the bit locations in the selected row are gated to sense amplifiers where the logic level of each cell is determined, latched and written back into the cell from which it was taken. $\overline{CAS}$ then activates the column decoders which select one of 64 semse amplifiers from each RAM chip and gates it to an output buffer. During refresh the interface logic will enable ROW Address lines only. The $\overline{RFSH}$ line when it becomes active, enables the ROW ADDRESS of the row to be refreshed.

Acess Time

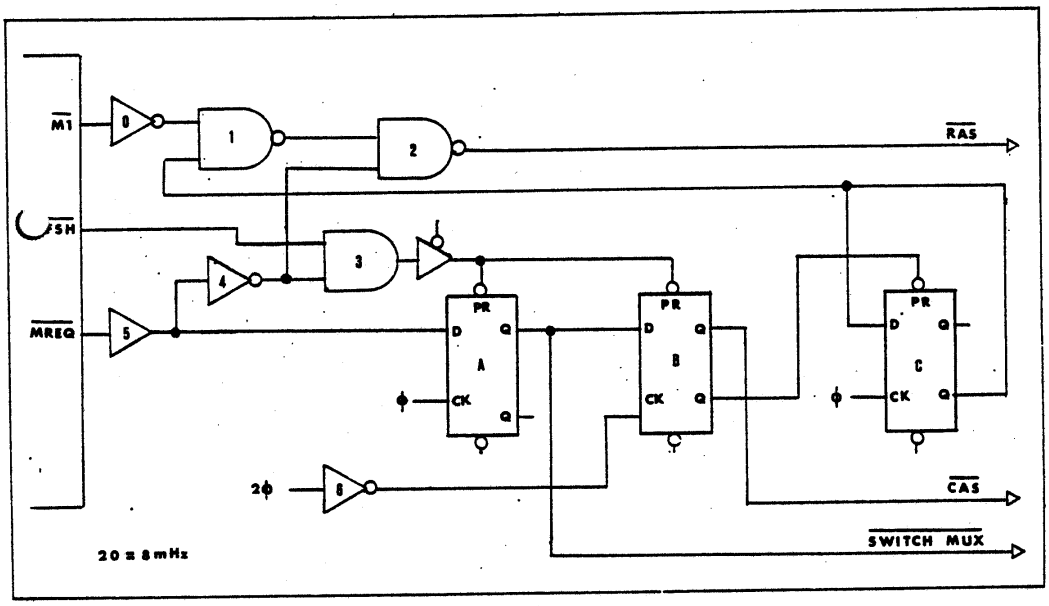
Most dynamic RAMs have access times which range from 150 ns. to 300 ns. Access time (tRAC) begins at the leading edge of the $\overline{RAS}$ strobe and ends when data from the memory location becomes available on the Data Bus. $\overline{CAS}$ is the critical parameter which terminates this cycle. The time interval between $\overline{RAS}$ and $\overline{CAS}$ is identified as the $\overline{RAS}$ to $\overline{CAS}$ delay time (tRCD). This delay is related to the worst case access time ($tRAC_{max}$) as illustrated below.



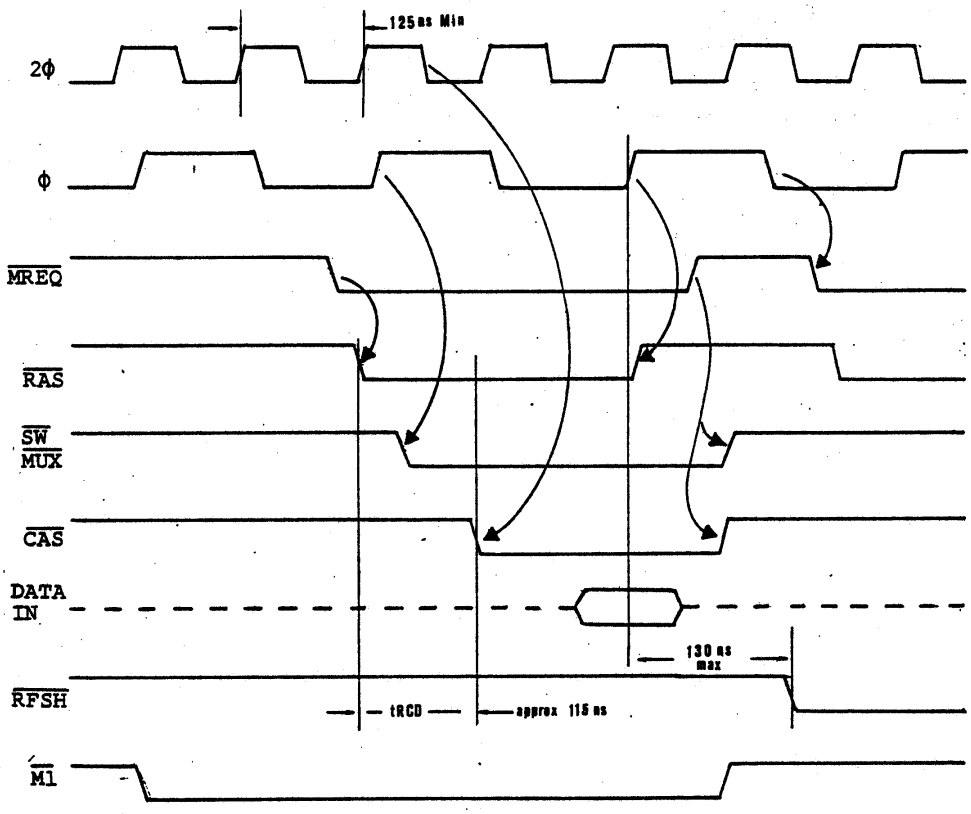
If the activation of $\overline{CAS}$ occurs beyond $tRCD_{max}$ then the total access time tRAC will exceed the worst case value $tRAC_{max}$, however, it should not occur before $tRCD_{min}$ because this is the minimum time within which Row Address Hold Time and Address Multiplexer change delay can occur. This simply means that $\overline{CAS}$ should be activated within the window provided by $tRCD_{max} - tRCD_{min}$ to obtain the minimum access time. This may not always be possible. One additional consideration is the delay from $\overline{MREQ}$ to $\overline{RAS}$; this should be as small as possible since it also affects the access time.

Dynamic Memory Control Signals

A control circuit is required to develop the three control strobes just discussed. The circuit must produce $\overline{RAS}$ synchronously with the system clock, followed almost immediately by $\overline{CAS}$. Since tRCD is on the order of 100 ns. or less, $\overline{CAS}$ cannot be derived directly from the clock. (At 4 mHz the clock pulse width is 125 ns.). Although one-shots might be used to to produce $\overline{CAS}$, to obtain precise delays of less than 100 ns. might be difficult. One simple approach to the problem would be to develop a stable clock signal at a frequency of 8 mHz or greater. The system clock (Ø) and the delayed strobe $\overline{CAS}$ could then be derived directly from the high frequency clock. The circuit shown employs this method.



Interface Timing

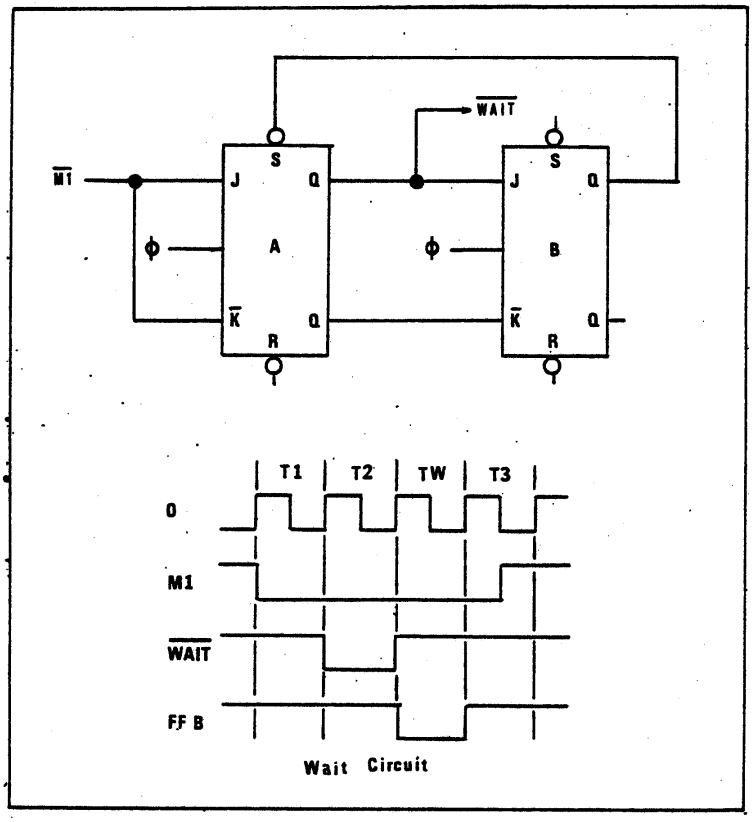


DYNAMIC RAM
INTERFACE TIMING

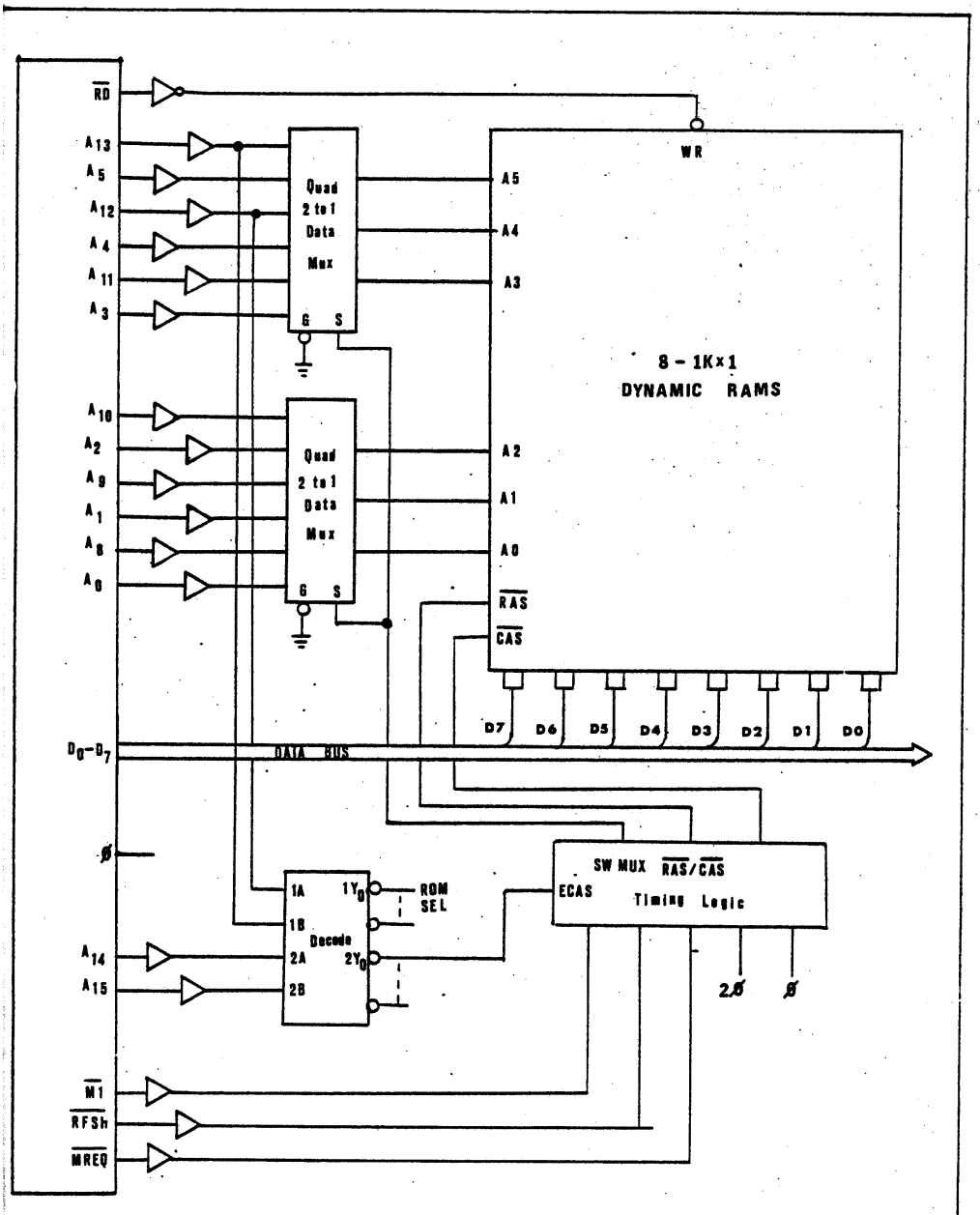
The interface timing diagram illustrates that $tRCD_{max}$ has been exceeded in this case. The access time measured from the leading edge of $\overline{CAS}$, (tCAC) is approximately 185 ns. This makes the total access time (tRAC) roughly 300 ns. By increasing the high speed clock to 16 mHz, tRCD could be decreased by approximately 50 ns.

Slow Memory Interface

Memory devices with long access times - slow static RAMS and EROMs such as the 2708 cannot respond to the CPU operating at 4 mHz without introducing Wait states. This problem must be considered in the overall design of the memory interface The simple circuit shown will provide one wait state (TW) during an Op Code Fetch cycle.



Wait Circuit


RD
A13
A5
A12
A4
A11
A3
Quad
2 to 1
Data
Mux
G
S
WR
A5
A4
A3
8 - 1K x 1
DYNAMIC RAMS
A10
A2
A9
A1
A8
A0
A2
A1
A0
RAS
CAS
D7
D6
D5
D4
D3
D2
D1
D0
D0-D7
DATA BUS
ø
1A
1B
2A
2B
1Y0
2Y0
Decode
ROM
SEL
A14
A15
SW MUX RAS/CAS
ECAS
Timing Logic
2ø
ø
M1
RFSh
MREQ

SECTION INT

Interrupts INT-1

Interrupt Interface INT-3

INTERRUPTS

Interrupts are needed to permit peripheral devices to temporarily suspend CPU operation in an ordered and systematic manner. During this interval, the CPU executes a peripheral service routine which may include the transfer a of data or status information and the initiation of prescribed control operations.

Z-80 Interrupt Structure

The Z-80 has two interrupt inputs; one provides a software maskable interrupt, and the other a non-maskable interrupt (NMI). The maskable interrupt (INT) can be selectively enabled or disabled by the programmer, the non-maskable interrupt (NMI) cannot.

There are two enable flip-flops in the Z-80 CPU. The first one called IFF_1 actually enables (or disables) the incoming interrupt. The second flip-flop, IFF_2 is used as temporary storage for IFF_1. Temporary storage of IFF is required during a non-maskable interrupt.

Interrupt Flip-Flop Operation

Action	IFF_1	IFF_2	Flag Op.
CPU Reset	0	0	----
DI	0	0	----
EI	1	1	----
LD A, I	.	.	$IFF_2 \rightarrow$ P Flag
LD A, R	.	.	" "
NMI Accepted	0	.	
RETN	IFF_2	.	$IFF_2 \rightarrow IFF_1$

Z-80 Interrupt Modes

Non-Maskable Interrupt

The NMI will be accepted at all times by the CPU. When the NMI occurs, the CPU does a restart to memory location 0066H. This is an automatic call to a memory location in Page 0.

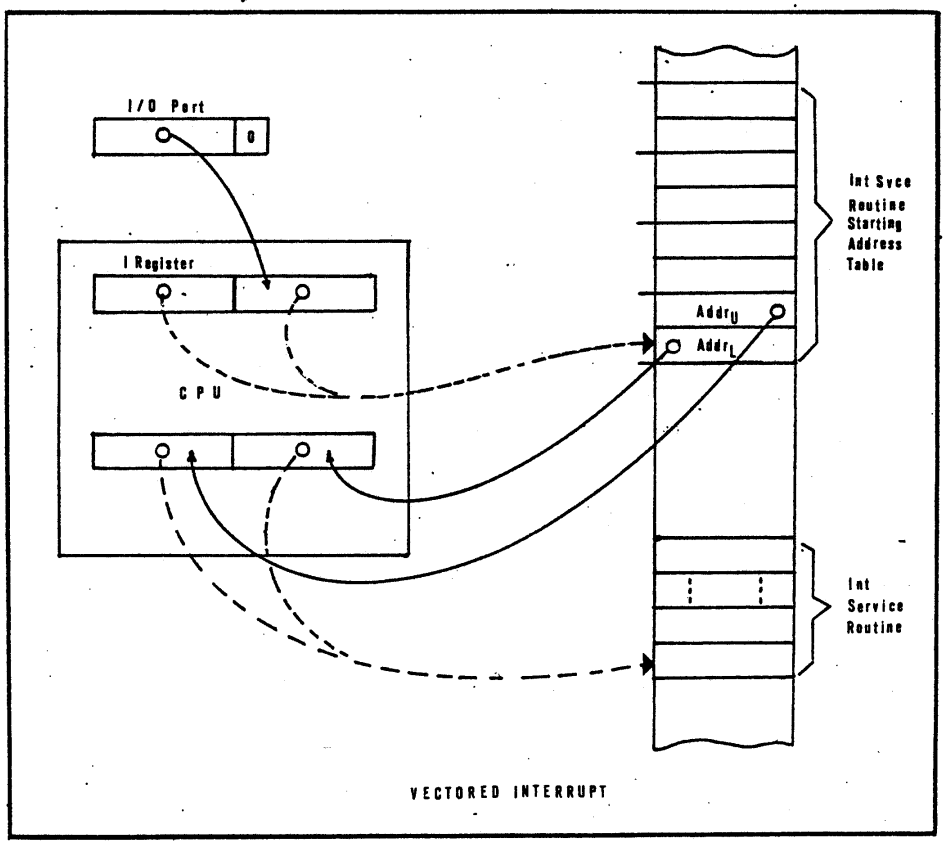
Maskable Interrupts

There are three modes of maskable interrupt:

Mode 0 - In this mode, the interrupting device can place any instruction on the Data Bus and the CPU will execute it. Normally, this mode is initiated either by a restart instruction RST p, which is a one byte call to one of 8 specified locations in page 0, or with a 3 byte call instruction.

Mode 1 - In this mode, the CPU responds to an interrupt by executing a restart to memory location 0038H.

Mode 2 - This mode is the most powerful of the interrupt modes. It is called the Vectored Interrupt Mode, because it provides an indirect call to any location in memory space, yet requires only a single 8 bit byte from the peripheral device. In order to use this mode, the programmer must maintain in memory a table of 16 bit starting address vectors; one for each interrupt service routine required. (Each starting address will occupy two adjacent locations in memory). This table may be located anywhere in memory. When an interrupt is accepted a 16 bit pointer will be formed, consisting of the upper 8 bits previously loaded to the Interrupt Vector Register I and the lower 8 bits provided by the peripheral device. This pointer identifies one of the starting Address Vectors in the table. The least significant bit of the pointer must always be zero because each starting address requires two memory locations.



VECTORED INTERRUPT

The lower address is put on the data bus in response to an interrupt acknowledge by the CPU.

INTERRUPT INTERFACE

There are three basic requirements for an interrupting I/O port:

A. The port must be able to initiate an interrupt.

B. The CPU must be able to identify the port which initiated the interrupt in order to provide the necessary interrupt service.

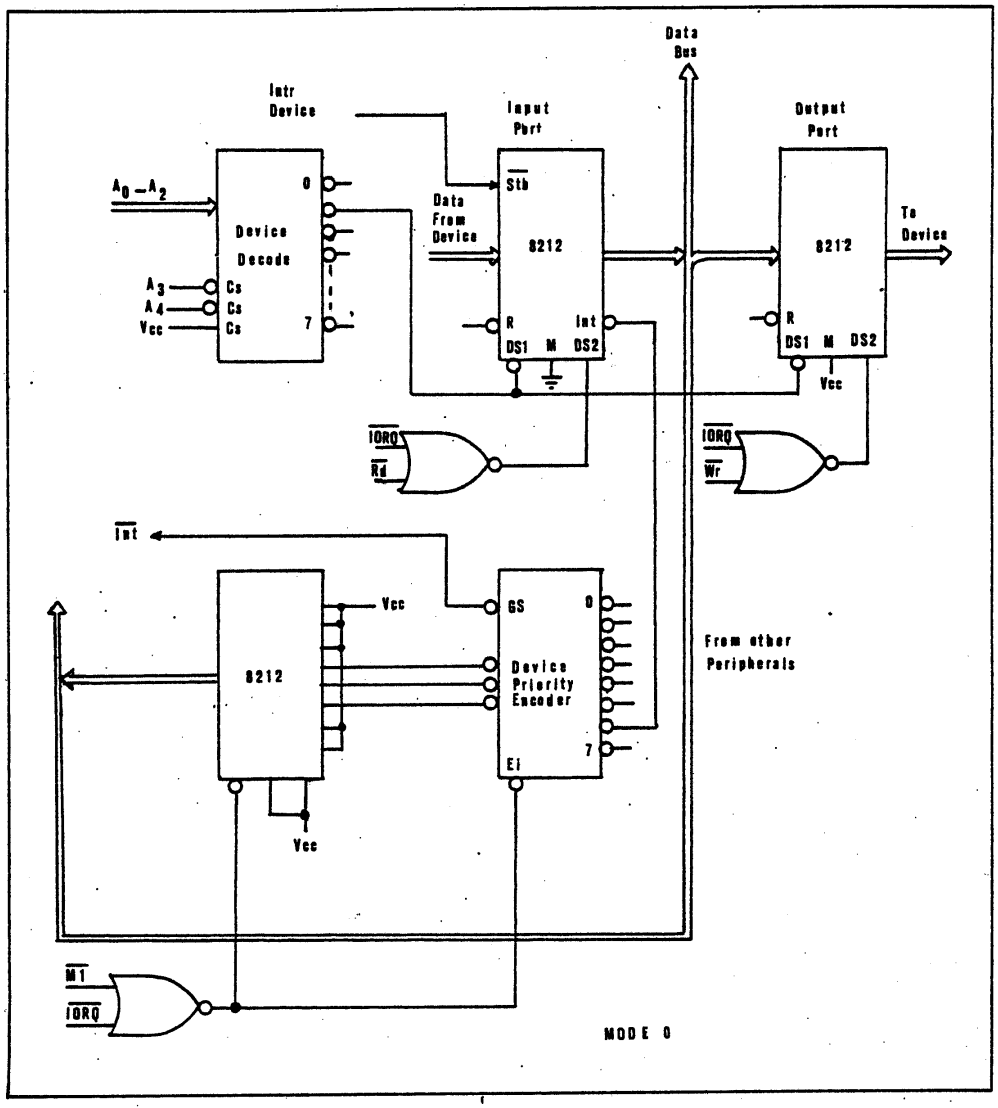
C. The port must be able to latch the data received if it is an output port, or place data on the data bus thru tri-state drivers if it is an input port.

The interrupt is automatically timed by the CPU. Data input and output, however, must be timed by the interface using the $\overline{IORQ}$, $\overline{MI}$, $\overline{Rd}$ and $\overline{WR}$ signals appropriately.

Mode 0 Interrupt

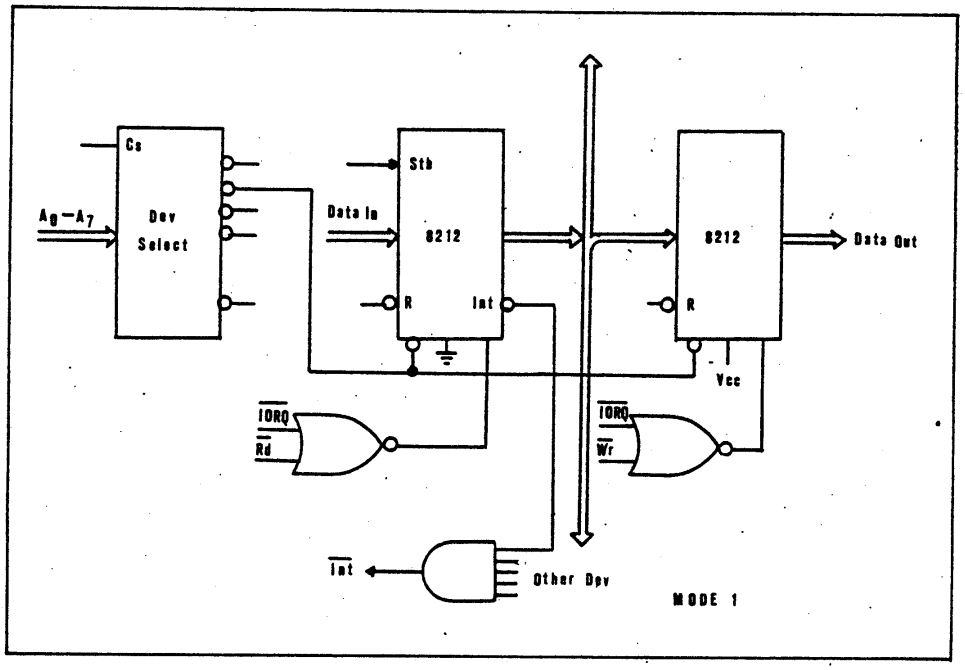
This interrupt mode is known as the 8080 mode. It requires either a RST p instruction or a 3 byte call instruction from the peripheral to initiate the interrupt service routine. The circuit illustrated below uses 8212 I/O ports for port latches and drivers and an 8205 8-to-1 decoder as an I/O device selector. In this circuit the input ports have interrupt capabilities, the outports do not. The circuitry could have been simplified considerably by using the Z-80 PIOs.

Mode-0 I/O Circuit



Mode 1 Interrupt

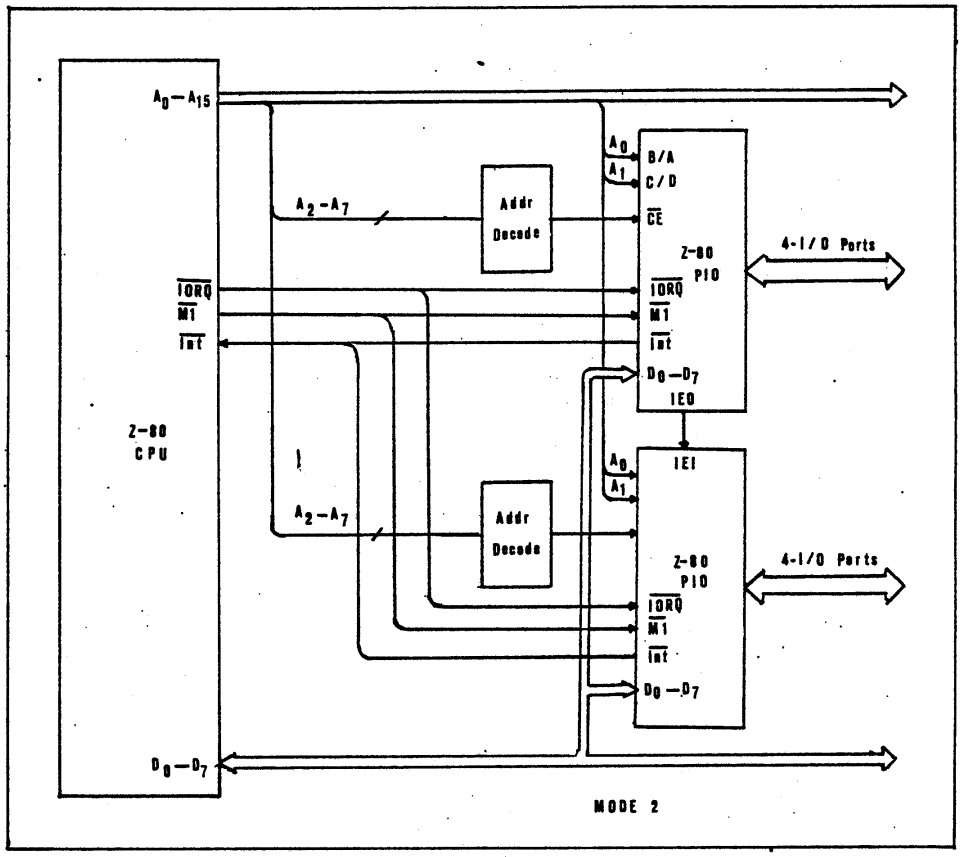
This interrupt mode causes the CPU to vector automatically to memory location 0038H, therefore no interrupt instruction is required from the interrupting port. The service routine, however, would have to poll the peripheral devices to determine which device required service.



Mode 2 Interrupt

This interrupt mode can be implemented with minimal hardware using the Z-80 PIO. This is illustrated on the following page.

8 Port I/O Interface for Mode 2 Interrupt



SECTION DMA

The Direct Memory Access Controller DMA-1

Z-80 DMA Pin-Out DMA-4

DMA Command Bytes DMA-5

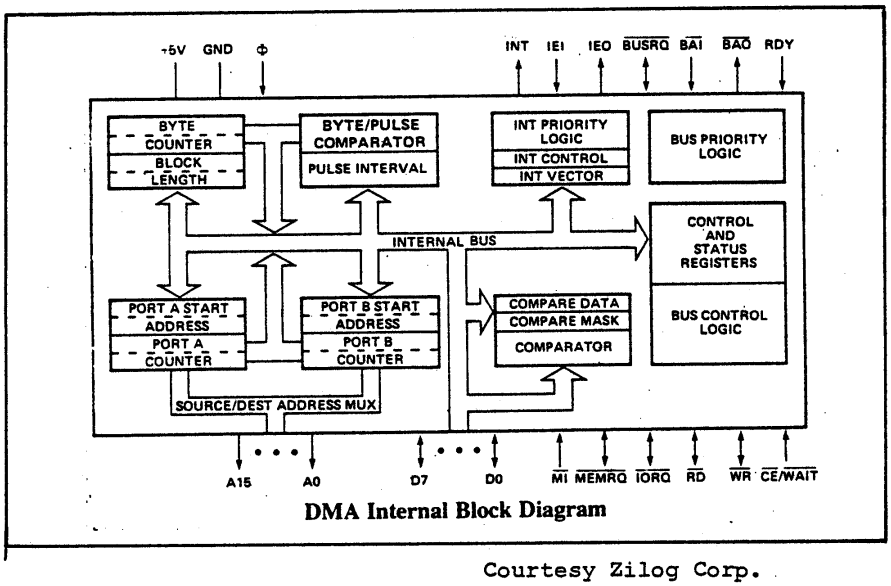
DMA Programming DMA-8

DIRECT MEMORY ACCESS CONTROLLER

The principle function of the Direct Memory Access Controller is to provide direct data transfers to and from the CPU working memory and the I/O device being controlled. It becomes more versatile if it can also provide direct transfers from one I/O device to another, and from one memory block to another. The Z-80 DMA has this capability.

The Z-80 DMA

The Z-80 DMA circuit is a programmable two port single channel device which provides all address, timing, and control signals required to transfer blocks of data between two ports within a Z-80 CPU system. The either or both of the two ports may be identified as Main Memory or any system peripheral.



DMA Internal Block Diagram

Courtesy Zilog Corp.

Internal Structure

Control Logic and Registers: Provide programmable control of class and mode of operation.

Address Byte Counter and Pulse Circuits: Generate the proper port addresses for read and write operations with provisions for incrementing or decrementing addresses. A zero flag is set in status register when byte counter content is zero. The pulse circuitry generates a pulse each time the byte counter lower 8 bits equal the contents of the pulse register.

Timing Circuits: Permit user to specify read/write timing for both channel ports.

Match Circuits: Hold match byte and mask byte which allow for a comparison with certain selected bits within a data byte.

$\overline{\text{INT}}$ and $\overline{\text{BUSRQ}}$ Circuits: Include a control register which specifies conditions under which DMA can generate an interrupt; priority encoding logic which selects an $\overline{\text{INT}}$ or $\overline{\text{BUSRQ}}$ output; also an interrupt vector for vectored interrupts.

Status Register: Maintains current state of DMA.

Register Description

Control Register: This register holds the DMA control information that determines mode and class of operation, and when to initiate a pulse or interrupt.

Timing Registers: Hold Read/Write timing parameters for the two I/O ports.

Interrupt Vector Register: Holds the 8 bit vector placed on the data bus by the DMA after receiving $\overline{\text{IORQ}}$ during an interrupt acknowledge sequence.

Block Length Register: Contains the total block length of data to be transferred or searched.

Byte Counter: Counts the number of bytes transferred (or searched). On a load or continue, the Byte Counter is reset to zero. Each byte transferred increments the register until it matches the contents of the Block Length Register. At this time the End of Block status flag is set, and the operation suspended if programmed to do so. Also, if programmed the DMA will generate an Interrupt.

Compare Register: Holds the byte for which match is being sought in a search operation.

Mask Register: Holds the 8 bit mask to determine which bits in the compare register are to be matched.

Starting Address Registers (Ports A and B): Hold the starting addresses for the ports involved in a Transfer Operation. (These are 16 bit addresses.) In a Search Operation, only one port address is required.

Address Counters (Ports A and B): These Counters are loaded with the contents of the corresponding Starting Address Register whenever Search or Transfer Operations begin. These counters are programmed to increment, decrement, or remain fixed.

Pulse Control Register: Holds program supplied length of block for which the DMA will provide a pulse on $\overline{INT}$ output. (This pulse will not generate an interrupt since both $\overline{BUSRQ}$ and $\overline{BUSAK}$ will be active.

Status Register: Contains the Status bits Match, End of Block, Ready Active, Interrupt Pending, and Write Address Valid. Status bits are valid when set.

DMA Memory and Peripheral Timing

The DMA timing is identical to the standard Z-80 timing for Memory Read and Write and I/O Read and Write, except when in the Variable Timing Mode.

Variable Timing Mode

The DMA can be programmed to increase the timing cycle from 3 time states to four or more, if wait states are introduced.

Modes of Operation

The DMA may be programmed for one of four modes of operation:

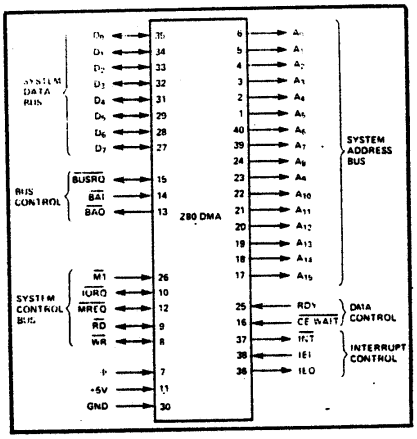
Byte at a time: Control is returned to the CPU after each one-byte cycle.

Burst: Operation continues as long as DMA RDY input is active. Control returns to the CPU when RDY is inactive or at end of block, or if match occurs, depending on the programming..

Continuous: The entire Search and/or Transfer of a block is completed before control is returned to the CPU.

Transparent: DMA operation occurs during normal refresh time without visible loss of CPU time.

Z-80 DMA Pin-Out



DMA Control Pin Description

Only those controls that are unique to the DMA are described in this section.

$\overline{RD}$ - Read to and from System Data Bus.

$\overline{WR}$ - Write to and from System Data Bus

$\overline{CE/Wait}$ -Chip Enable; may also be programmed as a $\overline{wait}$ during time the $\overline{BAI}$ is active.

$\overline{BUSRQ}$ - Requests Control of Z-80 System Buses.

$\overline{BAI}$ - Signals that System Buses have been released for DMA control.

$\overline{BAO}$ - Used to form (with $\overline{BAI}$) a daisy chain connection for system wide priority bus control.

RDY - A signal monitored by the DMA to determine when a peripheral device is ready for a read or write operation.

Programming the DMA

The DMA may be either in an "enable" or "disable" state. When it is in an "enable" state, it can gain control of the system buses for direct transactions between its ports. It is programmable

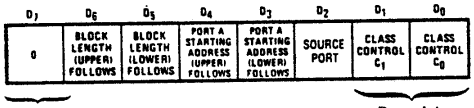
either in the "enable" or "disable" states. Programming automatically places the DMA in a "disable" until an enable command has been issued. The CPU programs the DMA by addressing it as an I/O port and sending a sequence of 8 bit commands via the system Data Bus, using output instructions. DMA transactions are initiated after the device has been programmed. When the DMA is powered up or reset by an means, it will be automatically placed in a "disable" state.

DMA Command Bytes

The command bytes contain information to be loaded into the DMA registers. The command structure is designed so that certain designated bits in some commands can be set to alert the DMA to expect the next byte to be written to a particular register.

There are six command bytes. Two of these are defined as Group 1 and contain the basic DMA set-up information. The other four are designated as Group 2 and specify additional detailed set-up information.

Command Byte 1A



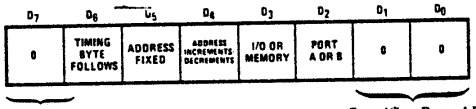
Specifies Group 1 — Byte 1A cannot be 00

C_1	C_0	Function
0	0	Not allowed. (Command Byte 1B)
0	1	Transfer Only.
1	0	Search Only.
1	1	Search and Transfer.

$D_2 = 1$ Port A is read from, Port B is written to (unless the Search Only Mode has been selected, in which case Port B is never addressed).

$D_2 = 0$ Port B is read from, Port A is written to (unless the Search Only Mode has been selected, in which case Port A is never addressed).

Command Byte 1B



Specifies Group 1 — Specifies Byte 1B

$D_4 = 1$ Address for this port increments after each byte.

$D_4 = 0$ Address for this port decrements after each byte.

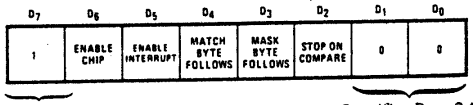
$D_3 = 1$ This port addresses an I/O peripheral.

$D_3 = 0$ This port addresses main memory.

$D_2 = 1$ This word programs Port A.

$D_2 = 0$ This word programs Port B.

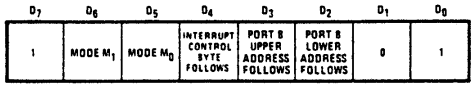
Command Byte 2A



Specifies Group 2 — Specifies Byte 2A

Courtesy Zilog Corp.

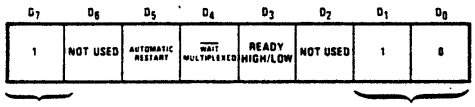
Command Byte 2B



Specifies Group 2 — Specifies Byte 2B

M_1	M_0	Mode
0	0	Byte
0	1	Continuous
1	0	Burst
1	1	Transparent

Command Byte 2C



Specifies Group 2 — Specifies Byte 2C

$D_5 = 1$ Automatically repeats entire operation when end of block is reached.

$D_5 = 0$ No affect.

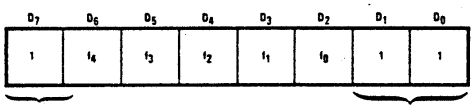
$D_4 = 1$ $\overline{CE}$ and $\overline{WAIT}$ multiplexed on same pin.

$D_4 = 0$ $\overline{CE}$ only.

$D_3 = 1$ Ready active high.

$D_3 = 0$ Ready active low.

Command Byte 2D



Specifies Group 2 — Specifies Byte 2D

Hex	f_4	f_3	f_2	f_1	f_0	
C3	1	0	0	0	0	Reset
C7	1	0	0	0	1	Reset Port A Timing
CB	1	0	0	1	0	Reset Port B Timing
CF	1	0	0	1	1	Load
D3	1	0	1	0	0	Continue
AB	0	1	0	1	0	Enable Int
AF	0	1	0	1	1	Disable Int
A3	0	1	0	0	0	Reset Int
87	0	0	0	0	1	Enable DMA
83	0	0	0	0	0	Disable DMA
BB	0	1	1	1	0	Read Byte Follows
A7	0	1	0	0	1	Reset RD
BF	0	1	1	1	1	RD Status
B3	0	1	1	0	0	Force Ready
B7	0	1	1	0	1	Enable After RETI
8B	0	0	0	1	0	Reset Status

Command Byte 2D Summary

Reset	Resets all interrupt circuitry, disables interrupts and bus req. logic.
Reset Timing A or B:	Resets timing for Port A or B to standard Z80-CPU timing.
Load:	Zeros Byte Counter and loads Starting Address for both Ports.
Continue:	Resets byte counter only. Addresses continue from present location.
Enable Interrupt:	Permits interrupt to occur.
Disable Interrupt:	Inhibits interrupt from occurring.
Reset Interrupt:	Resets and disables all interrupt circuits (similar to RETI).
Enable DMA. Disable DMA:	Overall enable or disable for all operations except interrupts; does not reset any functions.
Read Byte Follows:	Next write to DMA will contain a mask to program which readable registers are to be read.
Reset RD:	Next read will be from 1st register set as readable by response mask.
RD Status:	Next read will be from status register.
Force Ready:	Ready will be considered active regardless of the state of external RDY pin. Used for Mem-Mem operations where no RDY signal is needed.
Enable after RETI:	DMA will not request bus until after it has received a RETI.
RST Status:	Resets Match and End of Block status bits.

Read Byte

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
NOT USED	PORT B UPPER ADDR	PORT B LOWER ADDR	PORT A UPPER ADDR	PORT A LOWER ADDR	BYTE UPPER COUNT	BYTE LOWER COUNT	STATUS

A "1" in any bit position enables that register to be read.

Interrupt Control Byte

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
NO EFFECT	INTERRUPT BEFORE REQUESTING BUS	STATUS AFFECTS INTERRUPT VECTOR	INTERRUPT VECTOR FOLLOWS	PULSE COUNT FOLLOWS	PULSE GENERATED	INTERRUPT ON MATCH FOUND	INTERRUPT AT END OF BLOCK

A "1" in a bit position selects the option.

Timing Control Byte

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
$\overline{WR}$ END	$\overline{RD}$ END	NOT USED	NOT USED	$\overline{MREQ}$ END	$\overline{IORQ}$ END	T_1	T_0

T_1	T_0	Cycle Length
0	0	4
0	1	3
1	0	2
1	1	1

A "0" in D_2, D_3, D_6, or D_7 will cause the corresponding control signal to end ½ clock time before the end of the cycle. Note: the total operation (Read and Write in Transfer or Read in Search) must be at least 2 cycles long.

Mask Byte

A zero in a given bit position will cause a compare to be performed between that bit position in the compare word register and the same bit position in the data being read.

Match Byte

Up to an 8-bit word to be compared to $D_0 - D_7$ during a read. See MASK BYTE.

Status Byte (Status Bits Active–Low)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
NOT USED	NOT USED	END OF BLK	MATCH	INT. PENDING	NOT USED	READY ACTIVE	WRITE ADDRESS START

Pulse Count

This 8-bit word is loaded into a register. At the completion of each operation, the register is compared with the lower 8-bits of the byte counter. When it compares, the $\overline{INT}$ line is pulsed (but no interrupt is generated).

Interrupt Vector

This 8-bit byte is supplied to the CPU during Interrupt acknowledge if the DMA is the highest priority interrupting device.

If bit 5 of the Interrupt Control Byte (see p. 7) has been set and the DMA has been programmed to interrupt on a given status condition then D_1 and D_2 of the vector will be modified as follows:

Vector Bits	D_2	D_1	
	0	0	INT on RDY
	0	1	Match
	1	0	End of Blk
	1	1	Match, End of Blk

DMA Programming

The following example will illustrate how a DMA may be programmed to transfer a block of data from a peripheral device with a fixed address (Port A) to memory (Port B).

Port A	Data Flow	Port B
Peripheral Address	Block Length	Memory
0005H (Fixed)	1000H Bytes	Starting at 1050H

DMA Initialization Routine

```
0040    LD B, CH;       Load block length of Command Byte Table
0042    LD HL, 0050H;   Load starting address DMA Byte Table
0045    LD C, F1 H;     Load DMA Port Number
0047    OTIR,;          Load Command Byte Table to DMA
```

DMA Command Byte Table

Line	Code								Hex
50	0	1	1	0	1	1	0	1	6D
	Grp 1	Block Length Upper Lower Follows		Port A Hi Addr Lo None Follows		A→B	Transfer Mode Byte 1a		
51	0	0	0	0	0	1	0	1	05
	Port A Address Lower 8 Bits								
52	0	0	0	0	0	0	0	0	00
	Block Length Lower 8 Bits								
53	0	0	0	1	0	0	0	0	10
	Block Length Upper 8 Bits								
54	0	0	1	0	1	1	0	0	2C
	Grp 1	No Timing	Fixed Address	X	I/O Port	Port A	Byte 1b		
55	0	0	0	1	0	0	0	0	10
	Grp 1	No Timing	Addr. Changes	Addr. Incr.	Memory	Port B	Byte 1b		

56	1	1	0	0	1	1	0	1	CD
	Grp 2	Burst	Mode	No Int Ctr Byte	Port B Addr follows		Byte 2b		
57	0	1	0	1	0	0	0	0	50
	Port B Address Lower 8 Bits								
58	0	0	0	1	0	0	0	0	10
	Port B Address Upper 8 Bits								
59	1	0	0	0	1	0	1	0	8A
	Grp 2	X	No Auto RST	No Wait	RDY High	X	Byte 2c		
5A	1	1	0	0	1	1	1	1	CF
	Grp 2	Load Reset	Starting Block	Addr. Counter			Byte 2a		
5B	1	0	0	0	0	1	1	1	87
	Grp 2	Enable	DMA				Byte 2D		

SECTION APP

8-Bit Operation	APP-1
16-Bit Operation	APP-2
8-Bit Multiply Worksheet	APP-3
Multiply Program	APP-4
Controller Application	APP-5
20-Bit PN Sequence	APP-6
Random Number Generation	APP-7
Statistical Analysis Program	APP-8

N

8-Bit Operation

The following exercise in eight bit arithmetics can be used to demonstrate the z-80 capability to handle binary signed and unsigned numbers as well as decimal in both addition andsubtraction.

The program is assumed to be in R/W memory where it can easily be altered.

The two operands are on Port 0 and Port 1. Note the subtract assumes the quantity on Port 0 is the subtrahend.

The result (eight bits) is displayed on Port 3, while all the flags are displayed on Port 4.

The program is caused to loop continuously to allow easy changing of the operands to observe different results.

PROGRAM:

```
1000  DB 00          START:   IN    00H,A      ;GET X
1002  47                     LD    B,A        ;SAVE IT
1003  DB 01                   IN    01H,A      ;GET Y
1005  80      /90             ADD   B  /SUB  B
1006  00      /27             NOP     /DAA
1007  D3 03                   OUT   A,03H      ;DISPLAY RESULT
1009  F5                      PUSH  AF         ;GET FLAGS AND PUT THEM
100A  C1                      POP   BC         ;OUT ONTO PORT 4
100B  79                      LD    A,C
100C  D3 04                   OUT   A,04H
100E  C3 00 10                JP    START      ;REPEAT PROCESS
```

16-Bit Operation

The following exercise can be used to illustrate the Z-80 16-bit load, store, and arithemetics involving adding with a carry "in" equal to a one or zero; and subtracting with a borrow "in" equal to a one or zero. This can be done by starting the program at one of the two entry points: ADC0 or ADC1 respectively after writing some appropriate data into the operand spaces.

Note that the subtract with/without borrow in can be executed by changing the op code on line 101D from 4A to 4D.

It is assumed that the data and the program are in R/W memory and therefore easily changed.

DATA SPACE ASSIGNMENT: (PRESET BEFORE EXECUTING PROGRAM)

```
1000  XX          ;LSP OF X OPERAND
1001  XX          ;MSP OF X OPERAND, BOTH PRESET BEFORE
1002  YY          ;LSP OF Y OPERAND
1003  YY          ;MSP OF Y OPERAND
1004  SS          ;LSP OF 16-BIT RESULT
1005  SS          ;MSP OF 16-BIT RESULT
1006  00/01       ;FORMATTED CARRY OUT OR BORROW OUT OF RESULT
```

PROGRAM:

```
1010  37          ADC0:   SCF
1011  3F                  CCF             ;C=0
1012  18 01               JR ADCX
1014  37          ADC1:   SCF             ;C=1 ENTRY POINT
1015  ED 48 00 10 ADCX:   LD BC,(1000H)   ;GET X OPERAND
1019  2A 02 10            LD HL,(1002H)   ;GET Y OPERAND
101C  ED 4A               ADC HL,BC
101E  22 04 10            LD (1004H),HL   ;STORE RESULT
1021  3E 00               LD A,00H        ;FORMAT THE CARRY
1023  17                 RLA
1024  32 06 10            LD (1006H),A    ;STORE IT
1027  76                 HALT
```

If a subtraction is to be performed to see YYYY - XXXX - C, then alter line 101C as follows:

```
101C  ED 42               SBC HL,BC
```

Further, if a monitor program exists that permits easy scanning of memory data space, then instead of a halt, one may jump to that location to effect a memory read at the end of the program.

Controller Application

Z-80 Sequential Controller using the stack pointer to retrieve the time-code and display state from a look-up table.

PROGRAM:

```
10 00   31 00 11     START:    LD    SP, TABLE
10 03   C1           NEXT:     POP   BC
10 04   78                     LD    A,B
10 05   A0                     AND   B
10 06   28 F8                  JR    Z,F8H
10 08   79                     LD    A,C
10 09   D3 03                  OUT   3,A
10 0B   11 FF FF     CONT:     LD    DE,FFFFH
10 0E   21 FF FF               LD    HL,FFFFH
10 11   19           LOOP:     ADD   HL,DE
10 12   38 FD                  JR    C,FDH
10 14   10 F4                  DJNZ   ,F4H
10 16   18 EB                  JR     ,EBH
```

TABLE:

```
11 00   01           TABLE:    S1
11 01   01                     T1
11 02   0F                     S2
11 03   02                     T2
11 04   0A                     S3
11 05   F0                     T3
11 06   00                     00
11 07   00                     00                ;"END"
```

EIGHT-BIT MULTIPLY UNSIGNED NUMBERS

Exercise: Multiply two eight-bit unsigned numbers. Assume the multiplier is on PORT 01 and that the multiplicand is on PORT 02 The sixteen-bit result is to be displayed on PORTs 04, 03. Let the program loop continuously to allow easy change of the operands.

PROGRAM:

```
1000 00       START:  NOP  ;Place for "CF" -software interrupt.
1001 DB02             IN   A,02H        ;Get operands.
1003 6F               LD   L,A
1004 DB01             IN   A,01H
1006 67               LD   H,A
1007 0608      MULT:  LD   B,08H        ;Set count.
1009 AF               XOR  A
100A 110000           LD   DE,0000H     ;Clear product space.
100D CB0C      LOOP:  RRC  H            ;Test mier bit.
100F 3003             JR   NC,SHIFT
1011 7A         ADD:  LD   A,D
1012 85               ADD  L
1013 57               LD   D,A
1014 CB1A      SHIFT: RR   D
1016 CB1B             RR   E
1018 00               NOP  ;Place for "CF" -software interrupt.
1019 10F2             DJNZ LOOP
101B 7B      DISPLAY: LD   A,E          ;Output results.
101C D303             OUT  03H,A
101E 7A               LD   A,D
101F D304             OUT  04H,A
1021 18DD             JR   START
```

The software interrupts can be demonstrated by the instructor using the Monitor Program to observe the contents of the DE register pair at each partial product stage.

TALLY COUNTER PROGRAM

```
1000 010008      START:  LD   BC,0800H  ;Preset BIT CTR B, Tally C
1003 DB00                IN   A,00H     ;Get data
1005 17           TEST:  RLA            ;Tally number of 1's
1006 3001                JR   NC,ZERO
1008 0C            ONE:  INC  C
1009 10FA         ZERO:  DJNZ TEST
-------------------------------------------------------------------
100B 79                  LD   A,C       ;Compare to threshold.
100C FE04                CP   04H
100E 3E02                LD   A,02H     ;Equal condition.
1010 2806                JR   Z,DSPLY   ;#1's = #0's
1012 3E04                LD   A,04H
1014 3802                JR   C,DSPLY   ;#1's < #0's
1016 3E01                LD   A,01H     ;#1's > #0's
1018 D303         DSPLY: OUT  03H,A
101A 18E4                JR   START     ;Loop continuously.
```

20 Bit PN Sequence

The following is a twenty-bit Psuedo random number sequence, which is programmed with the alternate set of registers and therefore can be computed completely within the Z-80 CPU.

```
1000 D9          RANDX:  EXX   ;Select alternate register set.
1001 29                  ADD   HL,HL
1002 CB11                RL    C
1004 3E09                LD    A,09H
1006 A1                  AND   C
1007 EA0B10              JP    PE,EXIT   ;No relative test!
100A 23                  INC   HL
100B 7D          EXIT:   LD    A,L       ;Return number in A.
100C D9                  EXX
100D C9                  RET
```

Debounce Demonstration

Exercise to demonstrate the need for a debounce delay of a switch.

This program should be run first as written, to observe one action for each toggle of the switch B_0 of PORT 00.

PROGRAM:

```
1000 0103FF      START:  LD    BC,FF03H  ;B=counter, C=port pointer.
1003 04          ACTION: INC   B
1004 ED41                OUT   (C),B
1006 DB00        TEST0:  IN    A,00H
1008 1F                  RRA
1009 38FB                JR    C,TEST0   ;Wait for LO.
100B CD5000              CALL  50ms
100E DB00        TEST1:  IN    A,00H
1010 1F                  RRA
1011 30FB                JR    NC,TEST1  ;Wait for HI.
1013 CD5000              CALL  50ms
1016 18EB                JR    ACTION
```

The two delays should be NOPed to notice the effect. Are both delays needed? Does the delays slow the human reaction time?