PART 4
MICROPROCESSOR INSTRUCTION SET TABLES

EXPANDED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
		CPU Control Instructions							
8085	NOP	No OPeration	xx-x-x-x	4	1	Implied	00	nothing	Can be used to create time delays or leave extra spaces for instructions to be inserted at a later time.
Z80	NOP	No OPeration	xx-x-xxx						
8085	HLT	HALT	xx-x-x-x	5	1	Implied	76	stop processing	(8080 = 7 states)
Z80	HALT	HALT	xx-x-xxx	4					
		Data Transfer Instructions							
8085	MOV A,A	MOVe data to A from A	xx-x-x-x	4	1	Register	7F	A ← A	(8080 = 5 T states)
Z80	LD A,A	LoaD data into A from A	xx-x-xxx						
8085	MOV A,B	MOVe data to A from B	xx-x-x-x	4	1	Register	78	A ← B	(8080 = 5 T states)
Z80	LD A,B	LoaD data into A from B	xx-x-xxx						
8085	MOV A,C	MOVe data to A from C	xx-x-x-x	4	1	Register	79	A ← C	(8080 = 5 T states)
Z80	LD A,C	LoaD data into A from C	xx-x-xxx						
8085	MOV A,D	MOVe data to A from D	xx-x-x-x	4	1	Register	7A	A ← D	(8080 = 5 T states)
Z80	LD A,D	LoaD data into A from D	xx-x-xxx						
8085	MOV A,E	MOVe data to A from E	xx-x-x-x	4	1	Register	7B	A ← E	(8080 = 5 T states)
Z80	LD A,E	LoaD data into A from E	xx-x-xxx						

EXPANDED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY (*Continued*)

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	MOV A,H	MOVe data to A from H	xx-x-x-x	4	1	Register	7C	A ← H	(8080 = 5 T states)
Z80	LD A,H	LoaD data into A from H	xx-x-xxx						
8085	MOV A,L	MOVe data to A from L	xx-x-x-x	4	1	Register	7D	A ← L	(8080 = 5 T states)
Z80	LD A,L	LoaD data into A from L	xx-x-xxx						
8085	MOV A,M	MOVe data to A from M	xx-x-x-x	7	1	Reg Ind	7E	A ← M_{HL}	The data byte found at the memory location pointed to by the HL register pair is copied into the accumulator.
Z80	LD A,(HL)	LoaD data into A from (HL)	xx-x-xxx						
8085	MOV B,A	MOVe data to B from A	xx-x-x-x	4	1	Register	47	B ← A	(8080 = 5 T states)
Z80	LD B,A	LoaD data into B from A	xx-x-xxx						
8085	MOV B,B	MOVe data to B from B	xx-x-x-x	4	1	Register	40	B ← B	(8080 = 5 T states)
Z80	LD B,B	LoaD data into B from B	xx-x-xxx						
8085	MOV B,C	MOVe data to B from C	xx-x-x-x	4	1	Register	41	B ← C	(8080 = 5 T states)
Z80	LD B,C	LoaD data into B from C	xx-x-xxx						
8085	MOV B,D	MOVe data to B from D	xx-x-x-x	4	1	Register	42	B ← D	(8080 = 5 T states)
Z80	LD B,D	LoaD data into B from D	xx-x-xxx						
8085	MOV B,E	MOVe data to B from E	xx-x-x-x	4	1	Register	43	B ← E	(8080 = 5 T states)
Z80	LD B,E	LoaD data into B from E	xx-x-xxx						
8085	MOV B,H	MOVe data to B from H	xx-x-x-x	4	1	Register	44	B ← H	(8080 = 5 T states)
Z80	LD B,H	LoaD data into B from H	xx-x-xxx						

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	MOV B,L	MOVe data to B from L	xx-x-x-x	4	1	Register	45	B ← L	(8080 = 5 T states)
Z80	LD B,L	LoaD data into B from L	xx-x-xxx						
8085	MOV B,M	MOVe data to B from M_{HL}	xx-x-x-x	7	1	Reg Ind	46	B ← M_{HL}	The data byte found at the memory location pointed to by the HL register pair is copied into register B.
Z80	LD B,(HL)	LoaD data into B from (HL)	xx-x-xxx						
8085	MOV C,A	MOVe data to C from A	xx-x-x-x	4	1	Register	4F	C ← A	(8080 = 5 T states)
Z80	LD C,A	LoaD data into C from A	xx-x-xxx						
8085	MOV C,B	MOVe data to C from B	xx-x-x-x	4	1	Register	48	C ← B	(8080 = 5 T states)
Z80	LD C,B	LoaD data into C from B	xx-x-xxx						
8085	MOV C,C	MOVe data to C from C	xx-x-x-x	4	1	Register	49	C ← C	(8080 = 5 T states)
Z80	LD C,C	LoaD data into C from C	xx-x-xxx						
8085	MOV C,D	MOVe data to C from D	xx-x-x-x	4	1	Register	4A	C ← D	(8080 = 5 T states)
Z80	LD C,D	LoaD data into C from D	xx-x-xxx						
8085	MOV C,E	MOVe data to C from E	xx-x-x-x	4	1	Register	4B	C ← E	(8080 = 5 T states)
Z80	LD C,E	LoaD data into C from E	xx-x-xxx						
8085	MOV C,H	MOVe data to C from H	xx-x-x-x	4	1	Register	4C	C ← H	(8080 = 5 T states)
Z80	LD C,H	LoaD data into C from H	xx-x-xxx						
8085	MOV C,L	MOVe data to C from L	xx-x-x-x	4	1	Register	4D	C ← L	(8080 = 5 T states)
Z80	LD C,L	LoaD data into C from L	xx-x-xxx						

EXPANDED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	MOV C,M	MOVe data to **C** from M_{HL}	xx-x-x-x	7	1	Reg Ind	4E	C ← M_{HL}	The data byte found at the memory location pointed to by the HL register pair is copied into register C.
Z80	LD C,(HL)	LoaD data into **C** from **(HL)**	xx-x-xxx						
8085	MOV D,A	MOVe data to **D** from **A**	xx-x-x-x	4	1	Register	57	D ← A	(8080 = 5 T states)
Z80	LD D,A	LoaD data into **D** from **A**	xx-x-xxx						
8085	MOV D,B	MOVe data to **D** from **B**	xx-x-x-x	4	1	Register	50	D ← B	(8080 = 5 T states)
Z80	LD D,B	LoaD data into **D** from **B**	xx-x-xxx						
8085	MOV D,C	MOVe data to **D** from **C**	xx-x-x-x	4	1	Register	51	D ← C	(8080 = 5 T states)
Z80	LD D,C	LoaD data into **D** from **C**	xx-x-xxx						
8085	MOV D,D	MOVe data to **D** from **D**	xx-x-x-x	4	1	Register	52	D ← D	(8080 = 5 T states)
Z80	LD D,D	LoaD data into **D** from **D**	xx-x-xxx						
8085	MOV D,E	MOVe data to **D** from **E**	xx-x-x-x	4	1	Register	53	D ← E	(8080 = 5 T states)
Z80	LD D,E	LoaD data into **D** from **E**	xx-x-xxx						
8085	MOV D,H	MOVe data to **D** from **H**	xx-x-x-x	4	1	Register	54	D ← H	(8080 = 5 T states)
Z80	LD D,H	LoaD data into **D** from **H**	xx-x-xxx						
8085	MOV D,L	MOVe data to **D** from **L**	xx-x-x-x	4	1	Register	55	D ← L	(8080 = 5 T states)
Z80	LD D,L	LoaD data into **D** from **L**	xx-x-xxx						
8085	MOV D,M	MOVe data to **D** from M_{HL}	xx-x-x-x	7	1	Reg Ind	56	D ← M_{HL}	The data byte found at the memory location pointed to by the HL register pair is copied into register D.
Z80	LD D,(HL)	LoaD data into **D** from **(HL)**	xx-x-xxx						

Micro	Mnemonic	Operation	8085 > SZ-A-P-C Z80 > SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	MOV E,A	MOVe data to E from A	xx-x-x-x	4	1	Register	5F	E ← A	(8080 = 5 T states)
Z80	LD E,A	LoaD data into E from A	xx-x-xxx						
8085	MOV E,B	MOVe data to E from B	xx-x-x-x	4	1	Register	58	E ← B	(8080 = 5 T states)
Z80	LD E,B	LoaD data into E from B	xx-x-xxx						
8085	MOV E,C	MOVe data to E from C	xx-x-x-x	4	1	Register	59	E ← C	(8080 = 5 T states)
Z80	LD E,C	LoaD data into E from C	xx-x-xxx						
8085	MOV E,D	MOVe data to E from D	xx-x-x-x	4	1	Register	5A	E ← D	(8080 = 5 T states)
Z80	LD E,D	LoaD data into E from D	xx-x-xxx						
8085	MOV E,E	MOVe data to E from E	xx-x-x-x	4	1	Register	5B	E ← E	(8080 = 5 T states)
Z80	LD E,E	LoaD data into E from E	xx-x-xxx						
8085	MOV E,H	MOVe data to E from H	xx-x-x-x	4	1	Register	5C	E ← H	(8080 = 5 T states)
Z80	LD E,H	LoaD data into E from H	xx-x-xxx						
8085	MOV E,L	MOVe data to E from L	xx-x-x-x	4	1	Register	5D	E ← L	(8080 = 5 T states)
Z80	LD E,L	LoaD data into E from L	xx-x-xxx						
8085	MOV E,M	MOVe data to E from M_{HL}	xx-x-x-x	7	1	Reg Ind	5E	E ← M_{HL}	The data byte found at the memory location pointed to by the HL register pair is copied into register E.
Z80	LD E,(HL)	LoaD data into E from (HL)	xx-x-xxx						
8085	MOV H,A	MOVe data to H from A	xx-x-x-x	4	1	Register	67	H ← A	(8080 = 5 T states)
Z80	LD H,A	LoaD data into H from A	xx-x-xxx						

EXPANDED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Micro	Mnemonic	Operation	8085 > SZ-A-P-C Z80 > SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	MOV H,B	**MOVe** data to **H** from **B**	xx-x-x-x	4	1	Register	60	H ← B	(8080 = 5 T states)
Z80	LD H,B	**LoaD** data into **H** from **B**	xx-x-xxx						
8085	MOV H,C	**MOVe** data to **H** from **C**	xx-x-x-x	4	1	Register	61	H ← C	(8080 = 5 T states)
Z80	LD H,C	**LoaD** data into **H** from **C**	xx-x-xxx						
8085	MOV H,D	**MOVe** data to **H** from **D**	xx-x-x-x	4	1	Register	62	H ← D	(8080 = 5 T states)
Z80	LD H,D	**LoaD** data into **H** from **D**	xx-x-xxx						
8085	MOV H,E	**MOVe** data to **H** from **E**	xx-x-x-x	4	1	Register	63	H ← E	(8080 = 5 T states)
Z80	LD H,E	**LoaD** data into **H** from **E**	xx-x-xxx						
8085	MOV H,H	**MOVe** data to **H** from **H**	xx-x-x-x	4	1	Register	64	H ← H	(8080 = 5 T states)
Z80	LD H,H	**LoaD** data into **H** from **H**	xx-x-xxx						
8085	MOV H,L	**MOVe** data to **H** from **L**	xx-x-x-x	4	1	Register	65	H ← L	(8080 = 5 T states)
Z80	LD H,L	**LoaD** data into **H** from **L**	xx-x-xxx						
8085	MOV H,M	**MOVe** data to **H** from **M_{HL}**	xx-x-x-x	7	1	Reg Ind	66	H ← M_{HL}	The data byte found at the memory location pointed to by the HL register pair is copied into register H.
Z80	LD H,(HL)	**LoaD** data into **H** from **(HL)**	xx-x-xxx						
8085	MOV L,A	**MOVe** data to **L** from **A**	xx-x-x-x	4	1	Register	6F	L ← A	(8080 = 5 T states)
Z80	LD L,A	**LoaD** data into **L** from **A**	xx-x-xxx						
8085	MOV L,B	**MOVe** data to **L** from **B**	xx-x-x-x	4	1	Register	68	L ← B	(8080 = 5 T states)
Z80	LD L,B	**LoaD** data into **L** from **B**	xx-x-xxx						

Micro	Mnemonic	Operation	8085 > SZ-A-P-C Z80 > SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	MOV L,C	MOVe data to L from C	xx-x-x-x	4	1	Register	69	L ← C	(8080 = 5 T states)
Z80	LD L,C	LoaD data into L from C	xx-x-xxx						
8085	MOV L,D	MOVe data to L from D	xx-x-x-x	4	1	Register	6A	L ← D	(8080 = 5 T states)
Z80	LD L,D	LoaD data into L from D	xx-x-xxx						
8085	MOV L,E	MOVe data to L from E	xx-x-x-x	4	1	Register	6B	L ← E	(8080 = 5 T states)
Z80	LD L,E	LoaD data into L from E	xx-x-xxx						
8085	MOV L,H	MOVe data to L from H	xx-x-x-x	4	1	Register	6C	L ← H	(8080 = 5 T states)
Z80	LD L,H	LoaD data into L from H	xx-x-xxx						
8085	MOV L,L	MOVe data to L from L	xx-x-x-x	4	1	Register	6D	L ← L	(8080 = 5 T states)
Z80	LD L,L	LoaD data into L from L	xx-x-xxx						
8085	MOV L,M	MOVe data to L from M_{HL}	xx-x-x-x	7	1	Reg Ind	6E	L ← M_{HL}	The data byte found at the memory location pointed to by the HL register pair is copied into register L.
Z80	LD L,(HL)	LoaD data into L from (HL)	xx-x-xxx						
8085	MOV M,A	MOVe data to M_{HL} from A	xx-x-x-x	7	1	Reg Ind	77	M_{HL} ← A	The data in the accumulator is copied into the memory location pointed to by the HL register pair.
Z80	LD (HL),A	LoaD data into (HL) from A	xx-x-xxx						
8085	MOV M,B	MOVe data to M_{HL} from B	xx-x-x-x	7	1	Reg Ind	70	M_{HL} ← B	The data in register B is copied into the memory location pointed to by the HL register pair.
Z80	LD (HL),B	LoaD data into (HL) from B	xx-x-xxx						
8085	MOV M,C	MOVe data to M_{HL} from C	xx-x-x-x	7	1	Reg Ind	71	M_{HL} ← C	The data in register C is copied into the memory location pointed to by the HL register pair.
Z80	LD (HL),C	LoaD data into (HL) from C	xx-x-xxx						

EXPANDED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY (*Continued*)

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	MOV M,D	MOVe data to M_{HL} from D	xx-x-x-x	7	1	Reg Ind	72	$M_{HL} \leftarrow D$	The data in register D is copied into the memory location pointed to by the HL register pair.
Z80	LD (HL),D	LoaD data into (HL) from D	xx-x-xxx						
8085	MOV M,E	MOVe data to M_{HL} from E	xx-x-x-x	7	1	Reg Ind	73	$M_{HL} \leftarrow E$	The data in register E is copied into the memory location pointed to by the HL register pair.
Z80	LD (HL),E	LoaD data into (HL) from E	xx-x-xxx						
8085	MOV M,H	MOVe data to M_{HL} from H	xx-x-x-x	7	1	Reg Ind	74	$M_{HL} \leftarrow H$	The data in register H is copied into the memory location pointed to by the HL register pair.
Z80	LD (HL),H	LoaD data into (HL) from H	xx-x-xxx						
8085	MOV M,L	MOVe data to M_{HL} from L	xx-x-x-x	7	1	Reg Ind	75	$M_{HL} \leftarrow L$	The data in register L is copied into the memory location pointed to by the HL register pair.
Z80	LD (HL),L	LoaD data into (HL) from L	xx-x-xxx						
8085	MVI A,dd	MoVe Immediate dd to A	xx-x-x-x	7	2	Immed	3E	A ← dd	The data byte immediately following the op code is copied into the accumulator.
Z80	LD A,dd	LoaD dd into A	xx-x-xxx						
8085	MVI B,dd	MoVe Immediate dd to B	xx-x-x-x	7	2	Immed	06	B ← dd	The data byte immediately following the op code is copied into register B.
Z80	LD B,dd	LoaD dd into B	xx-x-xxx						
8085	MVI C,dd	MoVe Immediate dd to C	xx-x-x-x	7	2	Immed	0E	C ← dd	The data byte immediately following the op code is copied into register C.
Z80	LD C,dd	LoaD dd into C	xx-x-xxx						
8085	MVI D,dd	MoVe Immediate dd to D	xx-x-x-x	7	2	Immed	16	D ← dd	The data byte immediately following the op code is copied into register D.
Z80	LD D,dd	LoaD dd into D	xx-x-xxx						
8085	MVI E,dd	MoVe Immediate dd to E	xx-x-x-x	7	2	Immed	1E	E ← dd	The data byte immediately following the op code is copied into register E.
Z80	LD E,dd	LoaD dd into E	xx-x-xxx						
8085	MVI H,dd	MoVe Immediate dd to H	xx-x-x-x	7	2	Immed	26	H ← dd	The data byte immediately following the op code is copied into register H.
Z80	LD H,dd	LoaD dd into H	xx-x-xxx						

Micro	Mnemonic	Operation	8085 > SZ-A-P-C Z80 > SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085 Z80	MVI L,dd LD L,dd	MoVe Immediate dd to L LoaD dd into L	xx-x-x-x xx-x-xxx	7	2	Immed	2E	L ← dd	The data byte immediately following the op code is copied into register L.
8085 Z80	MVI M,dd LD (HL),dd	MoVe Immediate dd to M_{HL} LoaD dd into (HL)	xx-x-x-x xx-x-xxx	10	2	Immed/ Reg Ind	36	M_{HL} ← dd	The data byte immediately following the op code is copied into the memory location pointed to by the HL register pair.
8085 Z80	LXI B,dddd LD BC,dddd	Load eXtended Immediate dddd into register pair BC LoaD dddd into register pair BC	xx-x-x-x xx-x-xxx	10	3	Immed	01	BC ← dddd	Copy bytes 3 and 2 of the instruction into registers B and C respectively.
8085 Z80	LXI D,dddd LD DE,dddd	Load eXtended Immediate dddd into register pair DE LoaD dddd into register pair DE	xx-x-x-x xx-x-xxx	10	3	Immed	11	DE ← dddd	Copy bytes 3 and 2 of the instruction into registers D and E respectively.
8085 Z80	LXI H,dddd LD HL,dddd	Load eXtended Immediate dddd into register pair HL LoaD dddd into register pair HL	xx-x-x-x xx-x-xxx	10	3	Immed	21	HL ← dddd	Copy bytes 3 and 2 of the instruction into registers H and L respectively.
8085 Z80	LDAX B LD A,(BC)	LoaD Accumulator eXtended with data from mem loc BC LoaD Accumulator with data from mem loc (BC)	xx-x-x-x xx-x-xxx	7	1	Reg Ind	0A	A ← M_{BC}	Copy the data byte found at the memory location pointed to by the BC register pair into the accumulator.
8085 Z80	LDAX D LD A,(DE)	LoaD Accumulator eXtended with data from mem loc DE LoaD Accumulator with data from mem loc (DE)	xx-x-x-x xx-x-xxx	7	1	Reg Ind	1A	A ← M_{DE}	Copy the data byte found at the memory location pointed to by the DE register pair into the accumulator.
8085 Z80	LHLD aaaa LD HL,(aaaa)	Load HL Direct with data starting at aaaa LoaD HL with data starting at (aaaa)	xx-x-x-x xx-x-xxx	16	3	Direct	2A	L ← M_{aaaa} H ← M_{aaaa+1}	Copy the data byte found at memory location aaaa into the L register and the data byte found at the next memory location (aaaa+1) into the H register.

EXPANDED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	LDA aaaa	LoaD Accumulator with data from mem loc aaaa	xx-x-x-x	13	3	Direct	3A	$A \leftarrow M_{aaaa}$	Copy the contents of memory location aaaa into the Accumulator.
Z80	LD A,(aaaa)	LoaD Accumulator with data from mem loc (aaaa)	xx-x-xxx						
8085	STA aaaa	STore Accumulator in mem loc aaaa	xx-x-x-x	13	3	Direct	32	$M_{aaaa} \leftarrow A$	Copy the contents of the accumulator into memory location aaaa.
Z80	LD (aaaa),A	LoaD mem loc (aaaa) with the contents of the Accumulator	xx-x-xxx						
8085	STAX B	STore Accumulator eXtended at mem loc BC	xx-x-x-x	7	1	Reg Ind	02	$M_{BC} \leftarrow A$	Copy the contents of the accumulator into the memory location pointed to by the BC register pair.
Z80	LD (BC),A	LoaD mem loc (BC) with the contents of the Accumulator	xx-x-xxx						
8085	STAX D	STore Accumulator eXtended at mem loc DE	xx-x-x-x	7	1	Reg Ind	12	$M_{DE} \leftarrow A$	Copy the contents of the accumulator into the memory location pointed to by the DE register pair.
Z80	LD (DE),A	LoaD mem loc (DE) with the contents of the Accumulator	xx-x-xxx						
8085	SHLD aaaa	Store HL Direct at mem loc aaaa	xx-x-x-x	16	3	Direct	22	$M_{aaaa} \leftarrow L$ $M_{aaaa+1} \leftarrow H$	Copy the contents of register L into memory location aaaa and the contents of register H into the next (aaaa+1) memory location.
Z80	LD (aaaa),HL	LoaD mem loc starting at (aaaa) with contents of HL)	xx-x-xxx						
8085	XCHG	eXCHanGe DE with HL	xx-x-x-x	4	1	Register	EB	$DE \leftrightarrow HL$	Exchange the contents of the DE and HL register pairs.
Z80	EX DE,HL	EXchange DE with HL	xx-x-xxx						

Flag Instructions

8085	STC	SeT Carry flag	xx-x-x-1	4	1	Implied	37	$C \leftarrow 1$	The carry flag is normally designated as "CY" for the 8080/8085.
Z80	SCF	Set Carry Flag	xx-x-xx1						

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	CMC	CoMplement Carry flag	xx-x-x-C	4	1	Implied	3F	$C \leftarrow \overline{C}$	The carry flag is normally designated as "CY" for the 8080/8085.
Z80	CCF	Complement Carry Flag	xx-x-xxC						

Arithmetic Instructions

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	ADD A	**ADD A** to A	SZ-A-P-C	4	1	Register	87	$A \leftarrow A + A$	
Z80	ADD A,A	**ADD A** to **A**	SZ-H-P0C						
8085	ADD B	**ADD B** to A	SZ-A-P-C	4	1	Register	80	$A \leftarrow A + B$	
Z80	ADD A,B	**ADD B** to **A**	SZ-H-P0C						
8085	ADD C	**ADD C** to A	SZ-A-P-C	4	1	Register	81	$A \leftarrow A + C$	
Z80	ADD A,C	**ADD C** to **A**	SZ-H-P0C						
8085	ADD D	**ADD D** to A	SZ-A-P-C	4	1	Register	82	$A \leftarrow A + D$	
Z80	ADD A,D	**ADD D** to **A**	SZ-H-P0C						
8085	ADD E	**ADD E** to A	SZ-A-P-C	4	1	Register	83	$A \leftarrow A + E$	
Z80	ADD A,E	**ADD E** to **A**	SZ-H-P0C						
8085	ADD H	**ADD H** to A	SZ-A-P-C	4	1	Register	84	$A \leftarrow A + H$	
Z80	ADD A,H	**ADD H** to **A**	SZ-H-P0C						
8085	ADD L	**ADD L** to A	SZ-A-P-C	4	1	Register	85	$A \leftarrow A + L$	
Z80	ADD A,L	**ADD L** to **A**	SZ-H-P0C						
8085	ADD M	**ADD** M_{HL} to A	SZ-A-P-C	7	1	Reg Ind	86	$A \leftarrow A + M_{HL}$	Add the data byte whose memory location is pointed to by the HL register pair to the accumulator and store the results in the accumulator.
Z80	ADD A,(HL)	**ADD (HL)** to **A**	SZ-H-P0C						
8085	ADC A	AdD with Carry **A** to A	SZ-A-P-C	4	1	Register	8F	$A \leftarrow A + A + C$	The carry flag is usually designated by "CY" for the 8080/8085.
Z80	ADC A,A	AdD with Carry **A** to **A**	SZ-H-P0C						

EXPANDED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY (*Continued*)

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	ADC B	AdD with Carry B to A	SZ-A-P-C	4	1	Register	88	A ← A + B + C	The carry flag is usually designated by "CY" for the 8080/8085.
Z80	ADC A,B	AdD with Carry B to A	SZ-H-P0C						
8085	ADC C	AdD with Carry C to A	SZ-A-P-C	4	1	Register	89	A ← A + C + C	The carry flag is usually designated by "CY" for the 8080/8085.
Z80	ADC A,C	AdD with Carry C to A	SZ-H-P0C						
8085	ADC D	AdD with Carry D to A	SZ-A-P-C	4	1	Register	8A	A ← A + D + C	The carry flag is usually designated by "CY" for the 8080/8085.
Z80	ADC A,D	AdD with Carry D to A	SZ-H-P0C						
8085	ADC E	AdD with Carry E to A	SZ-A-P-C	4	1	Register	8B	A ← A + E + C	The carry flag is usually designated by "CY" for the 8080/8085.
Z80	ADC A,E	AdD with Carry E to A	SZ-H-P0C						
8085	ADC H	AdD with Carry H to A	SZ-A-P-C	4	1	Register	8C	A ← A + H + C	The carry flag is usually designated by "CY" for the 8080/8085.
Z80	ADC A,H	AdD with Carry H to A	SZ-H-P0C						
8085	ADC L	AdD with Carry L to A	SZ-A-P-C	4	1	Register	8D	A ← A + L + C	The carry flag is usually designated by "CY" for the 8080/8085.
Z80	ADC A,L	AdD with Carry L to A	SZ-H-P0C						
8085	ADC M	AdD with Carry M_{HL} to A	SZ-A-P-C	7	1	Reg Ind	8E	A ← A + M_{HL} + C	Add to the accumulator both the contents of the memory location pointed to by the HL register pair, and the carry flag, and then place this result in the accumulator.
Z80	ADC A,(HL)	AdD with Carry (HL) to A	SZ-H-P0C						
8085	SUB A	SUBtract A from A	SZ-A-P-C	4	1	Register	97	A ← A - A	
Z80	SUB A	SUBtract A from A	SZ-H-P1C						
8085	SUB B	SUBtract B from A	SZ-A-P-C	4	1	Register	90	A ← A - B	
Z80	SUB B	SUBtract B from A	SZ-H-P1C						
8085	SUB C	SUBtract C from A	SZ-A-P-C	4	1	Register	91	A ← A - C	
Z80	SUB C	SUBtract C from A	SZ-H-P1C						

Micro	Mnemonic	Operation	8085 > SZ-A-P-C Z80 > SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085 Z80	SUB D SUB D	SUBtract D from A SUBtract D from A	SZ-A-P-C SZ-H-P1C	4	1	Register	92	A ← A - D	
8085 Z80	SUB E SUB E	SUBtract E from A SUBtract E from A	SZ-A-P-C SZ-H-P1C	4	1	Register	93	A ← A - E	
8085 Z80	SUB H SUB H	SUBtract H from A SUBtract H from A	SZ-A-P-C SZ-H-P1C	4	1	Register	94	A ← A - H	
8085 Z80	SUB L SUB L	SUBtract L from A SUBtract L from A	SZ-A-P-C SZ-H-P1C	4	1	Register	95	A ← A - L	
8085 Z80	SUB M SUB (HL)	SUBtract M_{HL} from A SUBtract (HL) from A	SZ-A-P-C SZ-H-P1C	7	1	Reg Ind	96	A ← A - M_{HL}	Subtract the contents of the memory location pointed to by the HL register pair from the contents of the accumulator.
8085 Z80	SBB A SBC A,A	SuBtract with Borrow A from A SuBtract with Carry A from A	SZ-A-P-C SZ-H-P1C	4	1	Register	9F	A ← A - A - C	
8085 Z80	SBB B SBC A,B	SuBtract with Borrow B from A SuBtract with Carry B from A	SZ-A-P-C SZ-H-P1C	4	1	Register	98	A ← A - B - C	
8085 Z80	SBB C SBC A,C	SuBtract with Borrow C from A SuBtract with Carry C from A	SZ-A-P-C SZ-H-P1C	4	1	Register	99	A ← A - C - C	
8085 Z80	SBB D SBC A,D	SuBtract with Borrow D from A SuBtract with Carry D from A	SZ-A-P-C SZ-H-P1C	4	1	Register	9A	A ← A - D - C	
8085 Z80	SBB E SBC A,E	SuBtract with Borrow E from A SuBtract with Carry E from A	SZ-A-P-C SZ-H-P1C	4	1	Register	9B	A ← A - E - C	

EXPANDED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Micro	Mnemonic	Operation	8085 > SZ-A-P-C Z80 > SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085 Z80	SBB H SBC A,H	SuBtract with Borrow H from A SuBtract with Carry H from A	SZ-A-P-C SZ-H-P1C	4	1	Register	9C	A ← A - H - C	
8085 Z80	SBB L SBC A,L	SuBtract with Borrow L from A SuBtract with Carry L from A	SZ-A-P-C SZ-H-P1C	4	1	Register	9D	A ← A - L - C	
8085 Z80	SBB M SBC A,(HL)	SuBtract with Borrow M_{HL} from A SuBtract with Carry (HL) from A	SZ-A-P-C SZ-H-P1C	7	1	Reg Ind	9E	A ← A - M_{HL} - C	Subtract from the contents of the accumulator both the carry flag and the contents of the memory location pointed to by the HL register pair.
8085 Z80	DAD B ADD HL,BC	Double AdD BC to HL ADD BC to HL	xx-x-x-C xx-x-x0C	10 11	1	Register	09	HL ← HL + BC	
8085 Z80	DAD D ADD HL,DE	Double AdD DE to HL ADD DE to HL	xx-x-x-C xx-x-x0C	10 11	1	Register	19	HL ← HL + DE	
8085 Z80	DAD H ADD HL,HL	Double AdD HL to HL ADD HL to HL	xx-x-x-C xx-x-x0C	10 11	1	Register	29	HL ← HL + HL	
8085 Z80	ADI dd ADD A,dd	AdD Immediate dd to A ADD dd to A	SZ-A-P-C SZ-H-P0C	7	2	Immed	C6	A ← A + dd	
8085 Z80	ACI dd ADC A,dd	AdD with Carry Immediate dd to A AdD with Carry dd to A	SZ-A-P-C SZ-H-P0C	7	2	Immed	CE	A ← A + dd + C	
8085 Z80	SUI dd SUB dd	SUbtract Immediate dd from A SUBtract dd from A	SZ-A-P-C SZ-H-P1C	7	2	Immed	D6	A ← A - dd	

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	SBI dd	Subtract with Borrow Immediate dd from A	SZ-A-P-C	7	2	Immed	DE	A ← A - dd - C	
Z80	SBC A,dd	SuBtract with Carry dd from A	SZ-H-P1C						
8085	DAA	Decimal Adjust A	SZ-A-P-C	4	1	Implied	27	A ← BCD (A)	The 8-bit contents of the accumulator are adjusted to form two 4-bit binary-coded-decimal (BCD) digits.
Z80	DAA	Decimal Adjust A	SZ-H-PxC						

Logical Instructions

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	ANA A	ANd A with A	SZ-A-P-0	4	1	Register	A7	A ← A AND A	(8085) A flag=1
Z80	AND A	AND A with A	SZ-1-P00						(8080) A=ORing of bit 3 of the operands
8085	ANA B	ANd A with B	SZ-A-P-0	4	1	Register	A0	A ← A AND B	(8085) A flag=1
Z80	AND B	AND B with A	SZ-1-P00						(8080) A flag=ORing of bit 3 of the operands
8085	ANA C	ANd A with C	SZ-A-P-0	4	1	Register	A1	A ← A AND C	(8085) A flag=1
Z80	AND C	AND C with A	SZ-1-P00						(8080) A flag=ORing of bit 3 of the operands
8085	ANA D	ANd A with D	SZ-A-P-0	4	1	Register	A2	A ← A AND D	(8085) A flag=1
Z80	AND D	AND D with A	SZ-1-P00						(8080) A flag=ORing of bit 3 of the operands
8085	ANA E	ANd A with E	SZ-A-P-0	4	1	Register	A3	A ← A AND E	(8085) A flag=1
Z80	AND E	AND E with A	SZ-1-P00						(8080) A flag=ORing of bit 3 of the operands
8085	ANA H	ANd A with H	SZ-A-P-0	4	1	Register	A4	A ← A AND H	(8085) A flag=1
Z80	AND H	AND H with A	SZ-1-P00						(8080) A flag=ORing of bit 3 of the operands
8085	ANA L	ANd A with L	SZ-A-P-0	4	1	Register	A5	A ← A AND L	(8085) A flag=1
Z80	AND L	AND L with A	SZ-1-P00						(8080) A flag=ORing of bit 3 of the operands
8085	ANA M	ANd A with M_{HL}	SZ-A-P-0	7	1	Reg Ind	A6	A ← A AND M_{HL}	(8085) A flag=1
Z80	AND (HL)	AND (HL) with A	SZ-1-P00						(8080) A flag=ORing of bit 3 of the operands

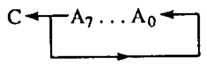
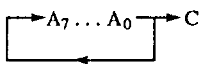
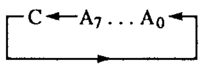
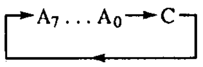
EXPANDED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY (*Continued*)

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	XRA A	eXclusively OR A with A	SZ-0-P-0	4	1	Register	AF	A ← A XOR A	
Z80	XOR A	eXclusively OR A with A	SZ-0-P00						
8085	XRA B	eXclusively OR A with B	SZ-0-P-0	4	1	Register	A8	A ← A XOR B	
Z80	XOR B	eXclusively OR A with B	SZ-0-P00						
8085	XRA C	eXclusively OR A with C	SZ-0-P-0	4	1	Register	A9	A ← A XOR C	
Z80	XOR C	eXclusively OR A with C	SZ-0-P00						
8085	XRA D	eXclusively OR A with D	SZ-0-P-0	4	1	Register	AA	A ← A XOR D	
Z80	XOR D	eXclusively OR A with D	SZ-0-P00						
8085	XRA E	eXclusively OR A with E	SZ-0-P-0	4	1	Register	AB	A ← A XOR E	
Z80	XOR E	eXclusively OR A with E	SZ-0-P00						
8085	XRA H	eXclusively OR A with H	SZ-0-P-0	4	1	Register	AC	A ← A XOR H	
Z80	XOR H	eXclusively OR A with H	SZ-0-P00						
8085	XRA L	eXclusively OR A with L	SZ-0-P-0	4	1	Register	AD	A ← A XOR L	
Z80	XOR L	eXclusively OR A with L	SZ-0-P00						
8085	XRA M	eXclusively OR A with M_{HL}	SZ-0-P-0	7	1	Reg Ind	AE	A ← A XOR M_{HL}	Exclusively OR the contents of the accumulator with the contents of the memory location pointed to by the HL register pair.
Z80	XOR (HL)	eXclusively OR A with (HL)	SZ-0-P00						
8085	ORA A	OR A with A	SZ-0-P-0	4	1	Register	B7	A ← A OR A	
Z80	OR A	OR A with A	SZ-0-P00						

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	ORA B	**OR A with B**	SZ-0-P-0	4	1	Register	B0	A ← A OR B	
Z80	OR B	**OR A with B**	SZ-0-P00						
8085	ORA C	**OR A with C**	SZ-0-P-0	4	1	Register	B1	A ← A OR C	
Z80	OR C	**OR A with C**	SZ-0-P00						
8085	ORA D	**OR A with D**	SZ-0-P-0	4	1	Register	B2	A ← A OR D	
Z80	OR D	**OR A with D**	SZ-0-P00						
8085	ORA E	**OR A with E**	SZ-0-P-0	4	1	Register	B3	A ← A OR E	
Z80	OR E	**OR A with E**	SZ-0-P00						
8085	ORA H	**OR A with H**	SZ-0-P-0	4	1	Register	B4	A ← A OR H	
Z80	OR H	**OR A with H**	SZ-0-P00						
8085	ORA L	**OR A with L**	SZ-0-P-0	4	1	Register	B5	A ← A OR L	
Z80	OR L	**OR A with L**	SZ-0-P00						
8085	ORA M	**OR A with M_{HL}**	SZ-0-P-0	7	1	Reg Ind	B6	A ← A OR M_{HL}	OR the contents of the accumulator with the contents of the memory location pointed to by the HL register pair.
Z80	OR (HL)	**OR A with (HL)**	SZ-0-P00						
8085	ANI dd	**AN**d **I**mmediate dd with A	SZ-A-P-0	7	2	Immed	E6	A ← A AND dd	(8085) A flag = 1 (8080) A flag = ORing of bit 3 of operands
Z80	AND dd	**AND** dd with A	SZ-1-P00						
8085	XRI dd	e**X**clusively **OR** **I**mmediate dd with A	SZ-0-P-0	7	2	Immed	EE	A ← A XOR dd	
Z80	XOR dd	e**X**clusively **OR** dd with A	SZ-0-P00						
8085	ORI dd	**OR** **I**mmediate dd with A	SZ-0-P-0	7	2	Immed	F6	A ← A OR dd	
Z80	OR dd	**OR** dd with A	SZ-0-P00						
8085	CMA	**C**o**M**plement **A**	xx-x-x-x	4	1	Implied	2F	A ← $\overline{A}$	Invert every bit in the accumulator. Form the 1's complement.
Z80	CPL	**C**om**PL**ement A	xx-1-x1x						

EXPANDED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Rotate and Shift Instructions

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	RLC	Rotate Left with Carry	xx-x-x-C	4	1	Implied	07		
Z80	RLCA	Rotate Left with Carry A	xx-0-x0C						
8085	RRC	Rotate Right with with Carry	xx-x-x-C	4	1	Implied	0F		
Z80	RRCA	Rotate Right with Carry A	xx-0-x0C						
8085	RAL	Rotate A Left	xx-x-x-C	4	1	Implied	17		
Z80	RLA	Rotate Left A	xx-0-x0C						
8085	RAR	Rotate A Right	xx-x-x-C	4	1	Implied	1F		
Z80	RRA	Rotate Right A	xx-0-x0C						

Increment and Decrement Instructions

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	INR A	INcRement A	SZ-A-P-x	4	1	Register	3C	$A \leftarrow A + 1$	(8080 = 5 states)
Z80	INC A	INCrement A	SZ-H-P0x						
8085	INR B	INcRement B	SZ-A-P-x	4	1	Register	04	$B \leftarrow B + 1$	(8080 = 5 states)
Z80	INC B	INCrement B	SZ-H-P0x						
8085	INR C	INcRement C	SZ-A-P-x	4	1	Register	0C	$C \leftarrow C + 1$	(8080 = 5 states)
Z80	INC C	INCrement C	SZ-H-P0x						
8085	INR D	INcRement D	SZ-A-P-x	4	1	Register	14	$D \leftarrow D + 1$	(8080 = 5 states)
Z80	INC D	INCrement D	SZ-H-P0x						
8085	INR E	INcRement E	SZ-A-P-x	4	1	Register	1C	$E \leftarrow E + 1$	(8080 = 5 states)
Z80	INC E	INCrement E	SZ-H-P0x						

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	INR H	**INcRement H**	SZ-A-P-x	4	1	Register	24	H ← H + 1	(8080 = 5 states)
Z80	INC H	**INCrement H**	SZ-H-P0x						
8085	INR L	**INcRement L**	SZ-A-P-x	4	1	Register	2C	L ← L + 1	(8080 = 5 states)
Z80	INC L	**INCrement L**	SZ-H-P0x						
8085	INR M	**INcRement M_{HL}**	SZ-A-P-x	10	1	Reg Ind	34	M_{HL} ← M_{HL} + 1	
Z80	INC (HL)	**INCrement (HL)**	SZ-H-P0x	11					
8085	INX B	**INcrement eXtended B**	xx-x-x-x	6	1	Register	03	BC ← BC + 1	(8080 = 5 states)
Z80	INC BC	**INCrement reg pair BC**	xx-x-xxx						
8085	INX D	**INcrement eXtended D**	xx-x-x-x	6	1	Register	13	DE ← DE + 1	(8080 = 5 states)
Z80	INC DE	**INCrement reg pair DE**	xx-x-xxx						
8085	INX H	**INcrement eXtended H**	xx-x-x-x	6	1	Register	23	HL ← HL + 1	(8080 = 5 states)
Z80	INC HL	**INCrement reg pair HL**	xx-x-xxx						
8085	DCR A	**DeCRement register A**	SZ-A-P-x	4	1	Register	3D	A ← A - 1	(8080 = 5 states)
Z80	DEC A	**DECrement register A**	SZ-H-P1x						
8085	DCR B	**DeCRement register B**	SZ-A-P-x	4	1	Register	05	B ← B - 1	(8080 = 5 states)
Z80	DEC B	**DECrement register B**	SZ-H-P1x						
8085	DCR C	**DeCRement register C**	SZ-A-P-x	4	1	Register	0D	C ← C - 1	(8080 = 5 states)
Z80	DEC C	**DECrement register C**	SZ-H-P1x						
8085	DCR D	**DeCRement register D**	SZ-A-P-x	4	1	Register	15	D ← D - 1	(8080 = 5 states)
Z80	DEC D	**DECrement register D**	SZ-H-P1x						
8085	DCR E	**DeCRement register E**	SZ-A-P-x	4	1	Register	1D	E ← E - 1	(8080 = 5 states)
Z80	DEC E	**DECrement register E**	SZ-H-P1x						

EXPANDED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY (*Continued*)

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	DCR H	DeCRement register H	SZ-A-P-x	4	1	Register	25	H ← H - 1	(8080 = 5 states)
Z80	DEC H	DECrement register H	SZ-H-P1x						
8085	DCR L	DeCRement register L	SZ-A-P-x	4	1	Register	2D	L ← L - 1	(8080 = 5 states)
Z80	DEC L	DECrement register L	SZ-H-P1x						
8085	DCR M	DeCRement M_{HL}	SZ-A-P-x	10	1	Reg Ind	35	$M_{HL} \leftarrow M_{HL} - 1$	
Z80	DEC (HL)	DECrement (HL)	SZ-H-P1x						
8085	DCX B	DeCrement eXtended register pair BC	xx-x-x-x	6	1	Register	0B	BC ← BC - 1	(8080 = 5 states)
Z80	DEC BC	DECrement register pair BC	xx-x-xxx						
8085	DCX D	DeCrement eXtended register pair DE	xx-x-x-x	6	1	Register	1B	DE ← DE - 1	(8080 = 5 states)
Z80	DEC DE	DECrement register pair DE	xx-x-xxx						
8085	DCX H	DeCrement eXtended register pair HL	xx-x-x-x	6	1	Register	2B	HL ← HL - 1	(8080 = 5 states)
Z80	DEC HL	DECrement register pair HL	xx-x-xxx						

Unconditional Jump Instructions

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	JMP aaaa	JuMP to mem loc aaaa	xx-x-x-x	10	3	Direct	C3	PC ← aaaa	
Z80	JP aaaa	JumP to mem loc aaaa	xx-x-xxx						
8085	PCHL	transfer to the Program Counter HL	xx-x-x-x	6	1	Register	E9	$PC_H \leftarrow H$	(8080 = 5 states) Transfer the contents of register H to the high-order byte of the program counter and the contents of register L to the low-order byte of the program counter.
Z80	JP (HL)	JumP to (HL)	xx-x-xxx	4				$PC_L \leftarrow L$	

Test (Compare) Instructions

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	CMP A	CoMPare A to A	SZ-A-P-C	4	1	Register	BF	A - A	If A = A then the Z flag = 1.
Z80	CP A	ComPare A to A	SZ-H-P1C						If A < A then the C flag = 1.
8085	CMP B	CoMPare B to A	SZ-A-P-C	4	1	Register	B8	A - B	If A = B then the Z flag = 1.
Z80	CP B	ComPare B to A	SZ-H-P1C						If A < B then the C flag = 1.
8085	CMP C	CoMPare C to A	SZ-A-P-C	4	1	Register	B9	A - C	If A = C then the Z flag = 1.
Z80	CP C	ComPare C to A	SZ-H-P1C						If A < C then the C flag = 1.
8085	CMP D	CoMPare D to A	SZ-A-P-C	4	1	Register	BA	A - D	If A = D then the Z flag = 1.
Z80	CP D	ComPare D to A	SZ-H-P1C						If A < D then the C flag = 1.
8085	CMP E	CoMPare E to A	SZ-A-P-C	4	1	Register	BB	A - E	If A = E then the Z flag = 1.
Z80	CP E	ComPare E to A	SZ-H-P1C						If A < E then the C flag = 1.
8085	CMP H	CoMPare H to A	SZ-A-P-C	4	1	Register	BC	A - H	If A = H then the Z flag = 1.
Z80	CP H	ComPare H to A	SZ-H-P1C						If A < H then the C flag = 1.
8085	CMP L	CoMPare L to A	SZ-A-P-C	4	1	Register	BD	A - L	If A = L then the Z flag = 1.
Z80	CP L	ComPare L to A	SZ-H-P1C						If A < L then the C flag = 1.
8085	CMP M	CoMPare M_{HL} to A	SZ-A-P-C	7	1	Reg Ind	BE	A - M_{HL}	If A = M_{HL} then the Z flag = 1.
Z80	CP (HL)	ComPare (HL) to A	SZ-H-P1C						If A < M_{HL} then the C flag = 1.
8085	CPI dd	ComPare Immediate dd to A	SZ-A-P-C	7	2	Immed	FE	A - dd	If A = dd then the Z flag = 1.
Z80	CP dd	ComPare dd to A	SZ-H-P1C						If A < dd then the C flag = 1.

EXPANDED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
		Conditional Jump (Branch) Instructions							
8085	JNZ aaaa	Jump if Not Zero to aaaa	xx-x-x-x	7/10	3	Direct	C2	PC ← aaaa	(8080 = 10 states)
Z80	JP NZ,aaaa	JumP if Not Zero to aaaa	xx-x-xxx	10				if Z = 0	PC_L ← byte 2 PC_H ← byte 3
8085	JZ aaaa	Jump if Zero to aaaa	xx-x-x-x	7/10	3	Direct	CA	PC ← aaaa	(8080 = 10 states)
Z80	JP Z,aaaa	JumP if Zero to aaaa	xx-x-xxx	10				if Z = 1	PC_L ← byte 2 PC_H ← byte 3
8085	JNC aaaa	Jump if No Carry to aaaa	xx-x-x-x	7/10	3	Direct	D2	PC ← aaaa	(8080 = 10 states)
Z80	JP NC,aaaa	JumP if No Carry to aaaa	xx-x-xxx	10				if C = 0	PC_L ← byte 2 PC_H ← byte 3
8085	JC aaaa	Jump if Carry to aaaa	xx-x-x-x	7/10	3	Direct	DA	PC ← aaaa	(8080 = 10 states)
Z80	JP C,aaaa	JumP if Carry to aaaa	xx-x-xxx	10				if C = 1	PC_L ← byte 2 PC_H ← byte 3
8085	JPO aaaa	Jump if Parity Odd to aaaa	xx-x-x-x	7/10	3	Direct	E2	PC ← aaaa	(8080 = 10 states)
Z80	JP PO,aaaa	JumP if Parity Odd to aaaa	xx-x-xxx	10				if P = 0	PC_L ← byte 2 PC_H ← byte 3
8085	JPE aaaa	Jump if Parity Even to aaaa	xx-x-x-x	7/10	3	Direct	EA	PC ← aaaa	(8080 = 10 states)
Z80	JP PE,aaaa	JumP if Parity Even to aaaa	xx-x-xxx	10				if P = 1	PC_L ← byte 2 PC_H ← byte 3
8085	JP aaaa	Jump if Plus to aaaa	xx-x-x-x	7/10	3	Direct	F2	PC ← aaaa	(8080 = 10 states)
Z80	JP P,aaaa	JumP if Plus to aaaa	xx-x-xxx	10				if S = 0	PC_L ← byte 2 PC_H ← byte 3
8085	JM aaaa	Jump if Minus to aaaa	xx-x-x-x	7/10	3	Direct	FA	PC ← aaaa	(8080 = 10 states)
Z80	JP M,aaaa	JumP if Minus to aaaa	xx-x-xxx	10				if S = 1	PC_L ← byte 2 PC_H ← byte 3
		Subroutine Instructions							
8085	CALL aaaa	CALL subroutine at aaaa	xx-x-x-x	18	3	Direct/ Reg Ind	CD	S ← PC_H	(8080 = 17 states)
Z80	CALL aaaa	CALL subroutine at aaaa	xx-x-xxx	17				S ← PC_L PC ← aaaa	The stack pointer is decremented as each new byte is pushed onto the stack. PC_H ← byte 3 PC_L ← byte 2

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	CNZ aaaa	Call if Not Zero subroutine at aaaa	xx-x-x-x	9/18	3	Direct/ Reg Ind	C4	if Z = 0 S ← PC_H S ← PC_L PC ← aaaa	(8080 = 11/17 states) The stack pointer is decremented as each new byte is pushed onto the stack. PC_H ← byte 3 PC_L ← byte 2
Z80	CALL NZ,aaaa	CALL if Not Zero subroutine at aaaa	xx-x-xxx	10/17					
8085	CZ aaaa	Call if Zero subroutine at aaaa	xx-x-x-x	9/18	3	Direct/ Reg Ind	CC	if Z = 1 S ← PC_H S ← PC_L PC ← aaaa	(8080 = 11/17 states) The stack pointer is decremented as each new byte is pushed onto the stack. PC_H ← byte 3 PC_L ← byte 2
Z80	CALL Z,aaaa	CALL if Zero subroutine at aaaa	xx-x-xxx	10/17					
8085	CNC aaaa	Call if No Carry subroutine at aaaa	xx-x-x-x	9/18	3	Direct/ Reg Ind	D4	if C = 0 S ← PC_H S ← PC_L PC ← aaaa	(8080 = 11/17 states) The stack pointer is decremented as each new byte is pushed onto the stack. PC_H ← byte 3 PC_L ← byte 2
Z80	CALL NC,aaaa	CALL if No Carry subroutine at aaaa	xx-x-xxx	10/17					
8085	CC aaaa	Call if Carry subroutine at aaaa	xx-x-x-x	9/18	3	Direct/ Reg Ind	DC	if C = 1 S ← PC_H S ← PC_L PC ← aaaa	(8080 = 11/17 states) The stack pointer is decremented as each new byte is pushed onto the stack. PC_H ← byte 3 PC_L ← byte 2
Z80	CALL C,aaaa	CALL if Carry subroutine at aaaa	xx-x-xxx	10/17					
8085	CPO aaaa	Call if Parity Odd subroutine at aaaa	xx-x-x-x	9/18	3	Direct/ Reg Ind	E4	if P = 0 S ← PC_H S ← PC_L PC ← aaaa	(8080 = 11/17 states) The stack pointer is decremented as each new byte is pushed onto the stack. PC_H ← byte 3 PC_L ← byte 2
Z80	CALL PO,aaaa	CALL if Parity Odd subroutine at aaaa	xx-x-xxx	10/17					
8085	CPE aaaa	Call if Parity Even subroutine at aaaa	xx-x-x-x	9/18	3	Direct/ Reg Ind	EC	if P = 1 S ← PC_H S ← PC_L PC ← aaaa	(8080 = 11/17 states) The stack pointer is decremented as each new byte is pushed onto the stack. PC_H ← byte 3 PC_L ← byte 2
Z80	CALL PE,aaaa	CALL if Parity Even subroutine at aaaa	xx-x-xxx	10/17					
8085	CP aaaa	Call if Plus subroutine at aaaa	xx-x-x-x	9/18	3	Direct/ Reg Ind	F4	if S = 0 S ← PC_H S ← PC_L PC ← aaaa	(8080 = 11/17 states) The stack pointer is decremented as each new byte is pushed onto the stack. PC_H ← byte 3 PC_L ← byte 2
Z80	CALL P,aaaa	CALL if Plus subroutine at aaaa	xx-x-xxx	10/17					

EXPANDED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY (*Continued*)

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	CM aaaa	Call if Minus subroutine at aaaa	xx-x-x-x	9/18	3	Direct/ Reg Ind	FC	if S = 1 S ← PC_H S ← PC_L PC ← aaaa	(8080 = 11/17 states) The stack pointer is decremented as each new byte is pushed onto the stack. PC_H ← byte 3 PC_L ← byte 2
Z80	CALL M,aaaa	CALL if Minus subroutine at aaaa	xx-x-xxx	10/17					
8085	RET	RETurn	xx-x-x-x	10	1	Reg Ind	C9	PC_L ← S PC_H ← S	The stack pointer is incremented as each byte is popped from the stack.
Z80	RET	RETurn	xx-x-xxx						
8085	RNZ	Return if Not Zero	xx-x-x-x	6/12	1	Reg Ind	C0	if Z = 0 PC_L ← S PC_H ← S	(8080 = 5/11 states) The stack pointer is incremented as each byte is popped from the stack.
Z80	RET NZ	RETurn if Not Zero	xx-x-xxx	5/10					
8085	RZ	Return if Zero	xx-x-x-x	6/12	1	Reg Ind	C8	if Z = 1 PC_L ← S PC_H ← S	(8080 = 5/11 states) The stack pointer is incremented as each byte is popped from the stack.
Z80	RET Z	RETurn if Zero	xx-x-xxx	5/10					
8085	RNC	Return if No Carry	xx-x-x-x	6/12	1	Reg Ind	D0	if C = 0 PC_L ← S PC_H ← S	(8080 = 5/11 states) The stack pointer is incremented as each byte is popped from the stack.
Z80	RET NC	RETurn if No Carry	xx-x-xxx	5/10					
8085	RC	Return if Carry	xx-x-x-x	6/12	1	Reg Ind	D8	if C = 1 PC_L ← S PC_H ← S	(8080 = 5/11 states) The stack pointer is incremented as each byte is popped from the stack.
Z80	RET C	RETurn if Carry	xx-x-xxx	5/10					
8085	RPO	Return if Parity Odd	xx-x-x-x	6/12	1	Reg Ind	E0	if P = 0 PC_L ← S PC_H ← S	(8080 = 5/11 states) The stack pointer is incremented as each byte is popped from the stack.
Z80	RET PO	RETurn if Parity Odd	xx-x-xxx	5/10					
8085	RPE	Return if Parity Even	xx-x-x-x	6/12	1	Reg Ind	E8	if P = 1 PC_L ← S PC_H ← S	(8080 = 5/11 states) The stack pointer is incremented as each byte is popped from the stack.
Z80	RET PE	RETurn if Parity Even	xx-x-xxx	5/10					

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	RP	Return if Plus	xx-x-x-x	6/12	1	Reg Ind	F0	if S = 0 $PC_L \leftarrow S$ $PC_H \leftarrow S$	(8080 = 5/11 states) The stack pointer is incremented as each byte is popped from the stack.
Z80	RET P	RETurn if Plus	xx-x-xxx	5/10					
8085	RM	Return if Minus	xx-x-x-x	6/12	1	Reg Ind	F8	if S = 1 $PC_L \leftarrow S$ $PC_H \leftarrow S$	(8080 = 5/11 states) The stack pointer is incremented as each byte is popped from the stack.
Z80	RET M	RETurn if Minus	xx-x-xxx	5/10					
8085	RST 0	ReStarT 0	xx-x-x-x	12	1	Reg Ind	C7	$S \leftarrow PC_H$ $S \leftarrow PC_L$ $PC \leftarrow 0000H$	(8080 = 11 states) The stack pointer is decremented as each new byte is pushed onto the stack.
Z80	RST 00H	ReStarT 00H	xx-x-xxx	11					
8085	RST 1	ReStarT 1	xx-x-x-x	12	1	Reg Ind	CF	$S \leftarrow PC_H$ $S \leftarrow PC_L$ $PC \leftarrow 0008H$	(8080 = 11 states) The stack pointer is decremented as each new byte is pushed onto the stack.
Z80	RST 08H	ReStarT 08H	xx-x-xxx	11					
8085	RST 2	ReStarT 2	xx-x-x-x	12	1	Reg Ind	D7	$S \leftarrow PC_H$ $S \leftarrow PC_L$ $PC \leftarrow 0010H$	(8080 = 11 states) The stack pointer is decremented as each new byte is pushed onto the stack.
Z80	RST 10H	ReStarT 10H	xx-x-xxx	11					
8085	RST 3	ReStarT 3	xx-x-x-x	12	1	Reg Ind	DF	$S \leftarrow PC_H$ $S \leftarrow PC_L$ $PC \leftarrow 0018H$	(8080 = 11 states) The stack pointer is decremented as each new byte is pushed onto the stack.
Z80	RST 18H	ReStarT 18H	xx-x-xxx	11					
8085	RST 4	ReStarT 4	xx-x-x-x	12	1	Reg Ind	E7	$S \leftarrow PC_H$ $S \leftarrow PC_L$ $PC \leftarrow 0020H$	(8080 = 11 states) The stack pointer is decremented as each new byte is pushed onto the stack.
Z80	RST 20H	ReStarT 20H	xx-x-xxx	11					
8085	RST 5	ReStarT 5	xx-x-x-x	12	1	Reg Ind	EF	$S \leftarrow PC_H$ $S \leftarrow PC_L$ $PC \leftarrow 0028H$	(8080 = 11 states) The stack pointer is decremented as each new byte is pushed onto the stack.
Z80	RST 28H	ReStarT 28H	xx-x-xxx	11					
8085	RST 6	ReStarT 6	xx-x-x-x	12	1	Reg Ind	F7	$S \leftarrow PC_H$ $S \leftarrow PC_L$ $PC \leftarrow 0030H$	(8080 = 11 states) The stack pointer is decremented as each new byte is pushed onto the stack.
Z80	RST 30H	ReStarT 30H	xx-x-xxx	11					

EXPANDED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	RST 7	ReStarT 7	xx-x-x-x	12	1	Reg Ind	FF	S ← PC_H S ← PC_L PC ← 0038H	(8080 = 11 states) The stack pointer is decremented as each new byte is pushed onto the stack.
Z80	RST 38H	ReStarT 38H	xx-x-xxx	11					

Stack Instructions

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	LXI SP,dddd	Load eXtended Immediate dddd into the Stack Pointer	xx-x-x-x	10	3	Immed	31	SP ← dddd	Copy bytes 3 and 2 of the instruction into the stack pointer.
Z80	LD SP,dddd	LoaD dddd into the Stack Pointer	xx-x-xxx						
8085	DAD SP	Double AdD SP to HL	xx-x-x-C	10	1	Register	39	HL ← HL + SP	
Z80	ADD HL,SP	ADD SP to HL	xx-x-x0C	11					
8085	INX SP	INcrement eXtended Stack Pointer	xx-x-x-x	6	1	Register	33	SP ← SP + 1	(8080 = 5 states)
Z80	INC SP	INCrement Stack Pointer	xx-x-xxx						
8085	DCX SP	DeCrement eXtended Stack Pointer	xx-x-x-x	6	1	Register	3B	SP ← SP - 1	(8080 = 5 states)
Z80	DEC SP	DECrement Stack Pointer	xx-x-xxx						
8085	PUSH B	PUSH reg pair BC	xx-x-x-x	12	1	Reg Ind	C5	S ← B S ← C	(8080 = 11 states) The stack pointer is decremented as each new byte is pushed onto the stack.
Z80	PUSH BC	PUSH reg pair BC	xx-x-xxx	11					
8085	PUSH D	PUSH reg pair DE	xx-x-x-x	12	1	Reg Ind	D5	S ← D S ← E	(8080 = 11 states) The stack pointer is decremented as each new byte is pushed onto the stack.
Z80	PUSH DE	PUSH reg pair DE	xx-x-xxx	11					
8085	PUSH H	PUSH reg pair HL	xx-x-x-x	12	1	Reg Ind	E5	S ← H S ← L	(8080 = 11 states) The stack pointer is decremented as each new byte is pushed onto the stack.
Z80	PUSH HL	PUSH reg pair HL	xx-x-xxx	11					

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	PUSH PSW	PUSH Processor Status Word	xx-x-x-x	12	1	Reg Ind	F5	S ← A S ← flags	(8080 = 11 states) The stack pointer is decremented as each new byte is pushed onto the stack. The "flags" byte is assembled in the normal order of the flags (8080/8085 = SZ-A-P-C and Z80 = SZ-H-PNC) for that microprocessor.
Z80	PUSH AF	PUSH Accumulator and Flags	xx-x-xxx	11					
8085	POP B	POP reg pair BC	xx-x-x-x	10	1	Reg Ind	C1	C ← S B ← S	The stack pointer is incremented as each byte is popped from the stack.
Z80	POP BC	POP reg pair BC	xx-x-xxx						
8085	POP D	POP reg pair DE	xx-x-x-x	10	1	Reg Ind	D1	E ← S D ← S	The stack pointer is incremented as each byte is popped from the stack.
Z80	POP DE	POP reg pair DE	xx-x-xxx						
8085	POP H	POP reg pair HL	xx-x-x-x	10	1	Reg Ind	E1	L ← S H ← S	The stack pointer is incremented as each byte is popped from the stack.
Z80	POP HL	POP reg pair HL	xx-x-xxx						
8085	POP PSW	POP Processor Status Word	SZ-A-P-C	10	1	Reg Ind	F1	flags ← S A ← S	The stack pointer is incremented as each byte is popped from the stack.
Z80	POP AF	POP Accumulator and Flag	SZ-H-PNC						
8085	XTHL	eXchange top of sTack with reg pair HL	xx-x-x-x	16	1	Reg Ind	E3	L ↔ S H ↔ $S_{(next)}$	(8080 = 18 states) Stack pointer does not change
Z80	EX (SP),HL	EXchange $M_{(SP)}$ with reg pair HL	xx-x-xxx	19					
8085	SPHL	move into SP the contents of reg pair HL	xx-x-x-x	6	1	Register	F9	SP ← HL	(8080 = 5 states)
Z80	LD SP,HL	LoaD into SP the contents of reg pair HL	xx-x-xxx						

Interrupt Instructions

Micro	Mnemonic	Operation	Flags	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	DI	Disable Interrupts	xx-x-x-x	4	1	Implied	F3	IFF ← 0	
Z80	DI	Disable Interrupts	xx-x-xxx						

EXPANDED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	EI	Enable Interrupts	xx-x-x-x	4	1	Implied	FB	IFF ← 1	
Z80	EI	Enable Interrupts	xx-x-xxx						
8085	RIM	(not covered here - see note at end of table)							
8085	SIM	(not covered here - see note at end of table)							

Input-Output Instructions

Micro	Mnemonic	Operation	8085>SZ-A-P-C Z80>SZ-H-PNC	T	#	Address Mode	Op	Boolean/Arith. Operation	Notes
8085	OUT dd	OUTput to port dd contents of A	xx-x-x-x	10	2	Direct	D3	dd port ← A	The contents of the accumulator are sent to a specified output port.
Z80	OUT dd,A	OUTput to port dd contents of A	xx-x-xxx	11					
8085	IN dd	INput into A one byte from port dd	xx-x-x-x	10	2	Direct	DB	A ← dd port (byte)	One byte from the specified port is copied into the accumulator.
Z80	IN A,dd	INput into A one byte from port dd	xx-x-xxx	11					

Address Modes

Implied
Register
Immediate
Direct
Register Indirect (Reg Ind)

Abbreviations and Explanations

PSW = program status word (flags)
S = stack
SP = stack pointer
PC = program counter
IFF = interrupt enable flip-flop
A = accumulator
B,C,D,E,H,L = registers
$_L$ = low-order byte
$_H$ = high-order byte
$A_7..A_0$ = accumulator bits 0 through 7

d = data (a single hex digit)
dd = data (two hex digits - 1 byte)
dddd = data (four hex digits - 2 bytes)

a = address (a single hex digit)
aa = address (two hex digits - 1 byte)
aaaa = address (four hex digits - 2 bytes)

Flags

If one of the flag letter designations is in the column for that particular flag it indicates that the flag is affected by this operation and could be set or cleared depending on the result of the operation. One of the following could also appear in a flag column:

-	=	no flag is represented by this column, a blank bit in the status register
x	=	flag not affected by this operation
1	=	flag always set by this operation
0	=	flag always cleared by this operation

8085

S = sign flag
Z = zero flag
A = auxiliary carry flag (usually labeled "AC")
P = parity flag
C = carry flag (usually labeled "CY")

Z80

S	=	sign
Z	=	zero flag
H	=	half carry flag
P	=	parity/overflow flag (usually labeled "P/V")
N	=	negative flag
C	=	carry flag

Symbols in the Page Heading

T	=	T states
#	=	number of bytes

Special Notes

States = When two numbers appear in the "States" column separated by a slash, the lower number indicates the number of states if the condition is false and the operation does not occur, and the larger number indicates the number of states if the condition is true and the operation does occur.

8080 = The 8080 behaves the same as the 8085 unless special information is provided in the "Notes" column for the 8080.

RIM & SIM = These two instructions related to interrupts are not covered in this table. They apply only to the 8085 (neither is available in either the 8080 or Z80).

Addressing Modes - A Summary

Implied: These instructions contain the source and destination of the data by implication.

Register: In this mode the operand and its source are specified and data is operated on in the registers only.

Immediate: The data to be operated on follows the instruction op code in memory; that is, it is the next byte in memory after the instruction.

Direct: The full address of the location of the operand in contained in bytes 2 and 3, that is, the next two bytes in memory after the instruction. The low-order byte comes first, and the high-order second.

Register Indirect (Reg Ind): In this addressing mode several steps are involved. Included in the instruction is a register pair; the contents of that register pair contains the address where that operand may be found, not the operand itself.

MINI TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY

CPU Control Instructions

8085	Z80	Op	Operation
NOP	NOP	00	Nothing happens
HLT	HALT	76	Stop processing

Data Transfer Instructions

8085	Z80	Op	Operation
MOV A,A	LD A,A	7F	A ← A
MOV A,B	LD A,B	78	A ← B
MOV A,C	LD A,C	79	A ← C
MOV A,D	LD A,D	7A	A ← D
MOV A,E	LD A,E	7B	A ← E
MOV A,H	LD A,H	7C	A ← H
MOV A,L	LD A,L	7D	A ← L
MOV A,M	LD A,(HL)	7E	A ← M_{HL}
MOV B,A	LD B,A	47	B ← A
MOV B,B	LD B,B	40	B ← B
MOV B,C	LD B,C	41	B ← C
MOV B,D	LD B,D	42	B ← D
MOV B,E	LD B,E	43	B ← E
MOV B,H	LD B,H	44	B ← H
MOV B,L	LD B,L	45	B ← L
MOV B,M	LD B,(HL)	46	B ← M_{HL}
MOV C,A	LD C,A	4F	C ← A
MOV C,B	LD C,B	48	C ← B
MOV C,C	LD C,C	49	C ← C
MOV C,D	LD C,D	4A	C ← D
MOV C,E	LD C,E	4B	C ← E
MOV C,H	LD C,H	4C	C ← H
MOV C,L	LD C,L	4D	C ← L
MOV C,M	LD C,(HL)	4E	C ← M_{HL}
MOV D,A	LD D,A	57	D ← A
MOV D,B	LD D,B	50	D ← B
MOV D,C	LD D,C	51	D ← C
MOV D,D	LD D,D	52	D ← D
MOV D,E	LD D,E	53	D ← E
MOV D,H	LD D,H	54	D ← H
MOV D,L	LD D,L	55	D ← L
MOV D,M	LD D,(HL)	56	D ← M_{HL}
MOV E,A	LD E,A	5F	E ← A
MOV E,B	LD E,B	58	E ← B
MOV E,C	LD E,C	59	E ← C
MOV E,D	LD E,D	5A	E ← D
MOV E,E	LD E,E	5B	E ← E
MOV E,H	LD E,H	5C	E ← H
MOV E,L	LD E,L	5D	E ← L
MOV E,M	LD E,(HL)	5E	E ← M_{HL}
MOV H,A	LD H,A	67	H ← A
MOV H,B	LD H,B	60	H ← B
MOV H,C	LD H,C	61	H ← C
MOV H,D	LD H,D	62	H ← D
MOV H,E	LD H,E	63	H ← E
MOV H,H	LD H,H	64	H ← H
MOV H,L	LD H,L	65	H ← L
MOV H,M	LD H,(HL)	66	H ← M_{HL}
MOV L,A	LD L,A	6F	L ← A
MOV L,B	LD L,B	68	L ← B
MOV L,C	LD L,C	69	L ← C

8085	Z80	Op	Operation
MOV L,D	LD L,D	6A	L ← D
MOV L,E	LD L,E	6B	L ← E
MOV L,H	LD L,H	6C	L ← H
MOV L,L	LD L,L	6D	L ← L
MOV L,M	LD L,(HL)	6E	L ← M_{HL}
MOV M,A	LD (HL),A	77	M_{HL} ← A
MOV M,B	LD (HL),B	70	M_{HL} ← B
MOV M,C	LD (HL),C	71	M_{HL} ← C
MOV M,D	LD (HL),D	72	M_{HL} ← D
MOV M,E	LD (HL),E	73	M_{HL} ← E
MOV M,H	LD (HL),H	74	M_{HL} ← H
MOV M,L	LD (HL),L	75	M_{HL} ← L
MVI A,dd	LD A,dd	3E	A ← dd
MVI B,dd	LD B,dd	06	B ← dd
MVI C,dd	LD C,dd	0E	C ← dd
MVI D,dd	LD D,dd	16	D ← dd
MVI E,dd	LD E,dd	1E	E ← dd
MVI H,dd	LD H,dd	26	H ← dd
MVI L,dd	LD L,dd	2E	L ← dd
MVI M,dd	LD (HL),dd	36	M_{HL} ← dd
LXI B,dddd	LD BC,dddd	01	BC ← dddd
LXI D,dddd	LD DE,dddd	11	DE ← dddd
LXI H,dddd	LD HL,dddd	21	HL ← dddd
LDAX B	LD A,(BC)	0A	A ← M_{BC}
LDAX D	LD A,(DE)	1A	A ← M_{DE}

LHLD aaaa	LD HL,(aaaa)	2A	L ← M_{aaaa}
			H ← M_{aaaa+1}

LDA aaaa	LD A,(aaaa)	3A	A ← M_{aaaa}
STA aaaa	LD (aaaa),A	32	M_{aaaa} ← A
STAX B	LD (BC),A	02	M_{BC} ← A

8085	Z80	Op	Operation
STAX D	LD (DE),A	12	M_{DE} ← A
SHLD aaaa	LD (aaaa),HL	22	M_{aaaa} ← L M_{aaaa+1} ← H
XCHG	EX DE,HL	EB	DE ↔ HL

Flag Instructions

STC	SCF	37	C ← 1
CMC	CCF	3F	C ← $\overline{C}$

Arithmetic Instructions

ADD A	ADD A,A	87	A ← A + A
ADD B	ADD A,B	80	A ← A + B
ADD C	ADD A,C	81	A ← A + C
ADD D	ADD A,D	82	A ← A + D
ADD E	ADD A,E	83	A ← A + E
ADD H	ADD A,H	84	A ← A + H
ADD L	ADD A,L	85	A ← A + L
ADD M	ADD A,(HL)	86	A ← A + M_{HL}
ADC A	ADC A,A	8F	A ← A + A + C
ADC B	ADC A,B	88	A ← A + B + C
ADC C	ADC A,C	89	A ← A + C + C
ADC D	ADC A,D	8A	A ← A + D + C
ADC E	ADC A,E	8B	A ← A + E + C
ADC H	ADC A,H	8C	A ← A + H + C
ADC L	ADC A,L	8D	A ← A + L + C
ADC M	ADC A,(HL)	8E	A ← A + M_{HL} + C
SUB A	SUB A	97	A ← A - A
SUB B	SUB B	90	A ← A - B
SUB C	SUB C	91	A ← A - C

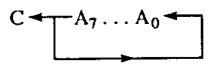
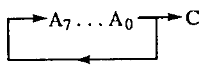
MINI TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY (*Continued*)

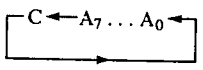
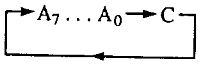
8085	Z80	Op	Operation
SUB D	SUB D	92	A ← A - D
SUB E	SUB E	93	A ← A - E
SUB H	SUB H	94	A ← A - H
SUB L	SUB L	95	A ← A - L
SUB M	SUB (HL)	96	A ← A - M_{HL}
SBB A	SBC A,A	9F	A ← A - A - C
SBB B	SBC A,B	98	A ← A - B - C
SBB C	SBC A,C	99	A ← A - C - C
SBB D	SBC A,D	9A	A ← A - D - C
SBB E	SBC A,E	9B	A ← A - E - C
SBB H	SBC A,H	9C	A ← A - H - C
SBB L	SBC A,L	9D	A ← A - L - C
SBB M	SBC A,(HL)	9E	A ← A - M_{HL} - C
DAD B	ADD HL,BC	09	HL ← HL + BC
DAD D	ADD HL,DE	19	HL ← HL + DE
DAD H	ADD HL,HL	29	HL ← HL + HL
ADI dd	ADD A,dd	C6	A ← A + dd
ACI dd	ADC A,dd	CE	A ← A + dd + C
SUI dd	SUB dd	D6	A ← A - dd
SBI dd	SBC A,dd	DE	A ← A - dd - C
DAA	DAA	27	A ← BCD (A)

Logical Instructions

8085	Z80	Op	Operation
ANA A	AND A	A7	A ← A AND A
ANA B	AND B	A0	A ← A AND B
ANA C	AND C	A1	A ← A AND C
ANA D	AND D	A2	A ← A AND D
ANA E	AND E	A3	A ← A AND E
ANA H	AND H	A4	A ← A AND H
ANA L	AND L	A5	A ← A AND L
ANA M	AND (HL)	A6	A ← A AND M_{HL}
XRA A	XOR A	AF	A ← A XOR A
XRA B	XOR B	A8	A ← A XOR B
XRA C	XOR C	A9	A ← A XOR C
XRA D	XOR D	AA	A ← A XOR D
XRA E	XOR E	AB	A ← A XOR E
XRA H	XOR H	AC	A ← A XOR H
XRA L	XOR L	AD	A ← A XOR L
XRA M	XOR (HL)	AE	A ← A XOR M_{HL}
ORA A	OR A	B7	A ← A OR A
ORA B	OR B	B0	A ← A OR B
ORA C	OR C	B1	A ← A OR C
ORA D	OR D	B2	A ← A OR D
ORA E	OR E	B3	A ← A OR E
ORA H	OR H	B4	A ← A OR H
ORA L	OR L	B5	A ← A OR L
ORA M	OR (HL)	B6	A ← A OR M_{HL}
ANI dd	AND dd	E6	A ← A AND dd
XRI dd	XOR dd	EE	A ← A XOR dd
ORI dd	OR dd	F6	A ← A OR dd
CMA	CPL	2F	A ← $\overline{A}$

Rotate and Shift Instructions

8085	Z80	Op	Operation
RLC	RLCA	07	
RRC	RRCA	0F	

8085	Z80	Op	Operation
RAL	RLA	17	
RAR	RRA	1F	

Increment and Decrement Instructions

8085	Z80	Op	Operation
INR A	INC A	3C	A ← A + 1
INR B	INC B	04	B ← B + 1
INR C	INC C	0C	C ← C + 1
INR D	INC D	14	D ← D + 1
INR E	INC E	1C	E ← E + 1
INR H	INC H	24	H ← H + 1
INR L	INC L	2C	L ← L + 1
INR M	INC (HL)	34	$M_{HL} \leftarrow M_{HL} + 1$
INX B	INC BC	03	BC ← BC + 1
INX D	INC DE	13	DE ← DE + 1
INX H	INC HL	23	HL ← HL + 1
DCR A	DEC A	3D	A ← A - 1
DCR B	DEC B	05	B ← B - 1
DCR C	DEC C	0D	C ← C - 1
DCR D	DEC D	15	D ← D - 1
DCR E	DEC E	1D	E ← E - 1
DCR H	DEC H	25	H ← H - 1
DCR L	DEC L	2D	L ← L - 1
DCR M	DEC (HL)	35	$M_{HL} \leftarrow M_{HL} - 1$
DCX B	DEC BC	0B	BC ← BC - 1
DCX D	DEC DE	1B	DE ← DE - 1
DCX H	DEC HL	2B	HL ← HL - 1

8085	Z80	Op	Operation

Unconditional Jump Instructions

8085	Z80	Op	Operation
JMP aaaa	JP aaaa	C3	PC ← aaaa
PCHL	JP (HL)	E9	$PC_H \leftarrow H$ $PC_L \leftarrow L$

Test (Compare) Instructions

8085	Z80	Op	Operation
CMP A	CP A	BF	A - A
CMP B	CP B	B8	A - B
CMP C	CP C	B9	A - C
CMP D	CP D	BA	A - D
CMP E	CP E	BB	A - E
CMP H	CP H	BC	A - H
CMP L	CP L	BD	A - L
CMP M	CP (HL)	BE	$A - M_{HL}$
CPI dd	CP dd	FE	A - dd

Conditional Jump (Branch) Instructions

8085	Z80	Op	Operation
JNZ aaaa	JP NZ,aaaa	C2	PC ← aaaa If Z = 0
JZ aaaa	JP Z,aaaa	CA	PC ← aaaa If Z = 1
JNC aaaa	JP NC,aaaa	D2	PC ← aaaa If C = 0
JC aaaa	JP C,aaaa	DA	PC ← aaaa If C = 1
JPO aaaa	JP PO,aaaa	E2	PC ← aaaa If P = 0
JPE aaaa	JP PE,aaaa	EA	PC ← aaaa If P = 1
JP aaaa	JP P,aaaa	F2	PC ← aaaa If S = 0

MINI TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY (*Continued*)

8085	Z80	Op	Operation
JM aaaa	JP M,aaaa	FA	PC ← aaaa If S = 1

Subroutine Instructions

8085	Z80	Op	Operation

CALL aaaa	CALL aaaa	CD	S ← PC_H
			S ← PC_L
			PC ← aaaa

CNZ aaaa	CALL NZ,aaaa	C4	If Z = 0
			S ← PC_H
			S ← PC_L
			PC ← aaaa

CZ aaaa	CALL Z,aaaa	CC	If Z = 1
			S ← PC_H
			S ← PC_L
			PC ← aaaa

CNC aaaa	CALL NC,aaaa	D4	If C = 0
			S ← PC_H
			S ← PC_L
			PC ← aaaa

CC aaaa	CALL C,aaaa	DC	If C = 1
			S ← PC_H
			S ← PC_L
			PC ← aaaa

CPO aaaa	CALL PO,aaaa	E4	If P = 0
			S ← PC_H
			S ← PC_L
			PC ← aaaa

CPE aaaa	CALL PE,aaaa	EC	If P = 1
			S ← PC_H
			S ← PC_L
			PC ← aaaa

CP aaaa	CALL P,aaaa	F4	If S = 0
			S ← PC_H
			S ← PC_L
			PC ← aaaa

CM aaaa	CALL M,aaaa	FC	If S = 1
			S ← PC_H
			S ← PC_L
			PC ← aaaa

RET	RET	C9	PC_L ← S PC_H ← S

8085	Z80	Op	Operation
RNZ	RET NZ	C0	If Z = 0 PC_L ← S PC_H ← S
RZ	RET Z	C8	If Z = 1 PC_L ← S PC_H ← S
RNC	RET NC	D0	If C = 0 PC_L ← S PC_H ← S
RC	RET C	D8	If C = 1 PC_L ← S PC_H ← S
RPO	RET PO	E0	If P = 0 PC_L ← S PC_H ← S
RPE	RET PE	E8	If P = 1 PC_L ← S PC_H ← S
RP	RET P	F0	If S = 0 PC_L ← S PC_H ← S
RM	RET M	F8	If S = 1 PC_L ← S PC_H ← S

RST 0	RST 00H	C7	S ← PC_H
			S ← PC_L
			PC ← 0000H

RST 1	RST 08H	CF	S ← PC_H
			S ← PC_L
			PC ← 0008H

RST 2	RST 10H	D7	S ← PC_H
			S ← PC_L
			PC ← 0010H

RST 3	RST 18H	DF	S ← PC_H
			S ← PC_L
			PC ← 0018H

RST 4	RST 20H	E7	S ← PC_H
			S ← PC_L
			PC ← 0020H

RST 5	RST 28H	EF	S ← PC_H
			S ← PC_L
			PC ← 0028H

8085	Z80	Op	Operation

RST 6	RST 30H	F7	S ← PC_H
			S ← PC_L
			PC ← 0030H

RST 7	RST 38H	FF	S ← PC_H
			S ← PC_L
			PC ← 0038H

Stack Instructions

8085	Z80	Op	Operation
LXI SP,dddd	LD SP,dddd	31	SP ← dddd
DAD SP	ADD HL,SP	39	HL ← HL + SP
INX SP	INC SP	33	SP ← SP + 1
DCX SP	DEC SP	3B	SP ← SP - 1

PUSH B	PUSH BC	C5	S ← B
			S ← C

PUSH D	PUSH DE	D5	S ← D
			S ← E

PUSH H	PUSH HL	E5	S ← H
			S ← L

PUSH PSW	PUSH AF	F5	S ← A
			S ← flags

POP B	POP BC	C1	C ← S
			B ← S

POP D	POP DE	D1	E ← S
			D ← S

POP H	POP HL	E1	L ← S
			H ← S

POP PSW	POP AF	F1	flags ← S
			A ← S

XTHL	EX (SP),HL	E3	L ↔ S H ↔ S (next)
SPHL	LD SP,HL	F9	SP ← HL

Interrupt Instructions

8085	Z80	Op	Operation
DI	DI	F3	IFF ← 0
EI	EI	FB	IFF ← 1

Input-Output Instructions

8085	Z80	Op	Operation
OUT dd	OUT dd,A	D3	dd port ← A
IN dd	IN A,dd	DB	A ← dd port (byte)

CONDENSED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY

CPU Control Instructions

8085	Z80	Op
NOP	NOP	00
HLT	HALT	76

Data Transfer Instructions

8085	Z80	Op
MOV A,A	LD A,A	7F
MOV A,B	LD A,B	78
MOV A,C	LD A,C	79
MOV A,D	LD A,D	7A
MOV A,E	LD A,E	7B
MOV A,H	LD A,H	7C
MOV A,L	LD A,L	7D
MOV A,M	LD A,(HL)	7E
MOV B,A	LD B,A	47
MOV B,B	LD B,B	40
MOV B,C	LD B,C	41
MOV B,D	LD B,D	42
MOV B,E	LD B,E	43
MOV B,H	LD B,H	44
MOV B,L	LD B,L	45
MOV B,M	LD B,(HL)	46
MOV C,A	LD C,A	4F
MOV C,B	LD C,B	48
MOV C,C	LD C,C	49
MOV C,D	LD C,D	4A
MOV C,E	LD C,E	4B
MOV C,H	LD C,H	4C
MOV C,L	LD C,L	4D
MOV C,M	LD C,(HL)	4E
MOV D,A	LD D,A	57
MOV D,B	LD D,B	50
MOV D,C	LD D,C	51
MOV D,D	LD D,D	52
MOV D,E	LD D,E	53
MOV D,H	LD D,H	54
MOV D,L	LD D,L	55
MOV D,M	LD D,(HL)	56
MOV E,A	LD E,A	5F
MOV E,B	LD E,B	58
MOV E,C	LD E,C	59
MOV E,D	LD E,D	5A
MOV E,E	LD E,E	5B

CONDENSED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY CATEGORY (*Continued*)

8085	Z80	Op
MOV E,H	LD E,H	5C
MOV E,L	LD E,L	5D
MOV E,M	LD E,(HL)	5E
MOV H,A	LD H,A	67
MOV H,B	LD H,B	60
MOV H,C	LD H,C	61
MOV H,D	LD H,D	62
MOV H,E	LD H,E	63
MOV H,H	LD H,H	64
MOV H,L	LD H,L	65
MOV H,M	LD H,(HL)	66
MOV L,A	LD L,A	6F
MOV L,B	LD L,B	68
MOV L,C	LD L,C	69
MOV L,D	LD L,D	6A
MOV L,E	LD L,E	6B
MOV L,H	LD L,H	6C
MOV L,L	LD L,L	6D
MOV L,M	LD L,(HL)	6E
MOV M,A	LD (HL),A	77
MOV M,B	LD (HL),B	70
MOV M,C	LD (HL),C	71
MOV M,D	LD (HL),D	72
MOV M,E	LD (HL),E	73
MOV M,H	LD (HL),H	74
MOV M,L	LD (HL),L	75
MVI A,dd	LD A,dd	3E
MVI B,dd	LD B,dd	06
MVI C,dd	LD C,dd	0E
MVI D,dd	LD D,dd	16
MVI E,dd	LD E,dd	1E
MVI H,dd	LD H,dd	26
MVI L,dd	LD L,dd	2E
MVI M,dd	LD (HL),dd	36
LXI B,dddd	LD BC,dddd	01
LXI D,dddd	LD DE,dddd	11
LXI H,dddd	LD HL,dddd	21
LDAX B	LD A,(BC)	0A
LDAX D	LD A,(DE)	1A
LHLD aaaa	LD HL,(aaaa)	2A
LDA aaaa	LD A,(aaaa)	3A
STA aaaa	LD (aaaa),A	32
STAX B	LD (BC),A	02
STAX D	LD (DE),A	12
SHLD aaaa	LD (aaaa),HL	22
XCHG	EX DE,HL	EB

Flag Instructions

8085	Z80	Op
STC	SCF	37
CMC	CCF	3F

Arithmetic Instructions

8085	Z80	Op
ADD A	ADD A,A	87
ADD B	ADD A,B	80
ADD C	ADD A,C	81
ADD D	ADD A,D	82
ADD E	ADD A,E	83
ADD H	ADD A,H	84
ADD L	ADD A,L	85
ADD M	ADD A,(HL)	86
ADC A	ADC A,A	8F
ADC B	ADC A,B	88
ADC C	ADC A,C	89
ADC D	ADC A,D	8A
ADC E	ADC A,E	8B
ADC H	ADC A,H	8C
ADC L	ADC A,L	8D
ADC M	ADC A,(HL)	8E
SUB A	SUB A	97
SUB B	SUB B	90
SUB C	SUB C	91
SUB D	SUB D	92
SUB E	SUB E	93
SUB H	SUB H	94
SUB L	SUB L	95
SUB M	SUB (HL)	96
SBB A	SBC A,A	9F
SBB B	SBC A,B	98
SBB C	SBC A,C	99
SBB D	SBC A,D	9A
SBB E	SBC A,E	9B
SBB H	SBC A,H	9C
SBB L	SBC A,L	9D
SBB M	SBC A,(HL)	9E
DAD B	ADD HL,BC	09
DAD D	ADD HL,DE	19
DAD H	ADD HL,HL	29
ADI dd	ADD A,dd	C6
ACI dd	ADC A,dd	CE
SUI dd	SUB dd	D6
SBI dd	SBC A,dd	DE
DAA	DAA	27

Logical Instructions

8085	Z80	Op
ANA A	AND A	A7
ANA B	AND B	A0
ANA C	AND C	A1
ANA D	AND D	A2
ANA E	AND E	A3
ANA H	AND H	A4
ANA L	AND L	A5
ANA M	AND (HL)	A6
XRA A	XOR A	AF
XRA B	XOR B	A8
XRA C	XOR C	A9
XRA D	XOR D	AA
XRA E	XOR E	AB
XRA H	XOR H	AC
XRA L	XOR L	AD
XRA M	XOR (HL)	AE
ORA A	OR A	B7
ORA B	OR B	B0
ORA C	OR C	B1
ORA D	OR D	B2
ORA E	OR E	B3
ORA H	OR H	B4
ORA L	OR L	B5
ORA M	OR (HL)	B6
ANI dd	AND dd	E6
XRI dd	XOR dd	EE
ORI dd	OR dd	F6
CMA	CPL	2F

Rotate and Shift Instructions

8085	Z80	Op
RLC	RLCA	07
RRC	RRCA	0F
RAL	RLA	17
RAR	RRA	1F

Increment and Decrement Instructions

8085	Z80	Op
INR A	INC A	3C
INR B	INC B	04
INR C	INC C	0C
INR D	INC D	14
INR E	INC E	1C
INR H	INC H	24
INR L	INC L	2C
INR M	INC (HL)	34
INX B	INC BC	03
INX D	INC DE	13
INX H	INC HL	23
DCR A	DEC A	3D
DCR B	DEC B	05
DCR C	DEC C	0D
DCR D	DEC D	15
DCR E	DEC E	1D
DCR H	DEC H	25

8085	Z80	Op
DCR L	DEC L	2D
DCR M	DEC (HL)	35
DCX B	DEC BC	0B
DCX D	DEC DE	1B
DCX H	DEC HL	2B

Unconditional Jump Instructions

JMP aaaa	JP aaaa	C3
PCHL	JP (HL)	E9

Test (Compare) Instructions

CMP A	CP A	BF
CMP B	CP B	B8
CMP C	CP C	B9
CMP D	CP D	BA
CMP E	CP E	BB
CMP H	CP H	BC
CMP L	CP L	BD
CMP M	CP (HL)	BE
CPI dd	CP dd	FE

Conditional Jump (Branch) Instructions

8085	Z80	Op
JNZ aaaa	JP NZ,aaaa	C2
JZ aaaa	JP Z,aaaa	CA
JNC aaaa	JP NC,aaaa	D2
JC aaaa	JP C,aaaa	DA
JPO aaaa	JP PO,aaaa	E2
JPE aaaa	JP PE,aaaa	EA
JP aaaa	JP P,aaaa	F2
JM aaaa	JP M,aaaa	FA

Subroutine Instructions

CALL aaaa	CALL aaaa	CD
CNZ aaaa	CALL NZ,aaaa	C4
CZ aaaa	CALL Z,aaaa	CC
CNC aaaa	CALL NC,aaaa	D4
CC aaaa	CALL C,aaaa	DC
CPO aaaa	CALL PO,aaaa	E4
CPE aaaa	CALL PE,aaaa	EC
CP aaaa	CALL P,aaaa	F4
CM aaaa	CALL M,aaaa	FC
RET	RET	C9
RNZ	RET NZ	C0
RZ	RET Z	C8
RNC	RET NC	D0
RC	RET C	D8
RPO	RET PO	E0
RPE	RET PE	E8
RP	RET P	F0
RM	RET M	F8
RST 0	RST 00H	C7
RST 1	RST 08H	CF
RST 2	RST 10H	D7
RST 3	RST 18H	DF
RST 4	RST 20H	E7
RST 5	RST 28H	EF
RST 6	RST 30H	F7
RST 7	RST 38H	FF

8085	Z80	Op

Stack Instructions

LXI SP,dddd	LD SP,dddd	31
DAD SP	ADD HL,SP	39
INX SP	INC SP	33
DCX SP	DEC SP	3B
PUSH B	PUSH BC	C5
PUSH D	PUSH DE	D5
PUSH H	PUSH HL	E5
PUSH PSW	PUSH AF	F5
POP B	POP BC	C1
POP D	POP DE	D1
POP H	POP HL	E1
POP PSW	POP AF	F1
XTHL	EX (SP),HL	E3
SPHL	LD SP,HL	F9

Interrupt Instructions

DI	DI	F3
EI	EI	FB

Input-Output Instructions

OUT dd	OUT dd,A	D3
IN dd	IN A,dd	DB

CONDENSED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY OP CODE

Op	8080/8085	Z80
00	NOP	NOP
01	LXI B,dddd	LD BC,dddd
02	STAX B	LD (BC),A
03	INX B	INC BC
04	INR B	INC B
05	DCR B	DEC B
06	MVI B,dd	LD B,dd
07	RLC	RLCA
09	DAD B	ADD HL,BC
0A	LDAX B	LD A,(BC)
0B	DCX B	DEC BC
0C	INR C	INC C
0D	DCR C	DEC C
0E	MVI C,dd	LD C,dd
0F	RRC	RRCA
11	LXI D,dddd	LD DE,dddd
12	STAX D	LD (DE),A
13	INX D	INC DE
14	INR D	INC D
15	DCR D	DEC D
16	MVI D,dd	LD D,dd
17	RAL	RLA
19	DAD D	ADD HL,DE
1A	LDAX D	LD A,(DE)
1B	DCX D	DEC DE
1C	INR E	INC E
1D	DCR E	DEC E
1E	MVI E,dd	LD E,dd
1F	RAR	RRA
21	LXI H,dddd	LD HL,dddd
22	SHLD aaaa	LD (aaaa),HL
23	INX H	INC HL
24	INR H	INC H

CONDENSED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED BY OP CODE (*Continued*)

Op	8080/8085	Z80	Op	8080/8085	Z80	Op	8080/8085	Z80
25	DCR H	DEC H	5F	MOV E,A	LD E,A	96	SUB M	SUB (HL)
26	MVI H,dd	LD H,dd	60	MOV H,B	LD H,B	97	SUB A	SUB A
27	DAA	DAA	61	MOV H,C	LD H,C	98	SBB B	SBC A,B
29	DAD H	ADD HL,HL	62	MOV H,D	LD H,D	99	SBB C	SBC A,C
2A	LHLD aaaa	LD HL,(aaaa)	63	MOV H,E	LD H,E	9A	SBB D	SBC A,D
2B	DCX H	DEC HL	64	MOV H,H	LD H,H	9B	SBB E	SBC A,E
2C	INR L	INC L	65	MOV H,L	LD H,L	9C	SBB H	SBC A,H
2D	DCR L	DEC L	66	MOV H,M	LD H,(HL)	9D	SBB L	SBC A,L
2E	MVI L,dd	LD L,dd	67	MOV H,A	LD H,A	9E	SBB M	SBC A,(HL)
2F	CMA	CPL	68	MOV L,B	LD L,B	9F	SBB A	SBC A,A
31	LXI SP,dddd	LD SP,dddd	69	MOV L,C	LD L,C	A0	ANA B	AND B
32	STA aaaa	LD (aaaa),A	6A	MOV L,D	LD L,D	A1	ANA C	AND C
33	INX SP	INC SP	6B	MOV L,E	LD L,E	A2	ANA D	AND D
34	INR M	INC (HL)	6C	MOV L,H	LD L,H	A3	ANA E	AND E
35	DCR M	DEC (HL)	6D	MOV L,L	LD L,L	A4	ANA H	AND H
36	MVI M,dd	LD (HL),dd	6E	MOV L,M	LD L,(HL)	A5	ANA L	AND L
37	STC	SCF	6F	MOV L,A	LD L,A	A6	ANA M	AND (HL)
39	DAD SP	ADD HL,SP	70	MOV M,B	LD (HL),B	A7	ANA A	AND A
3A	LDA aaaa	LD A,(aaaa)	71	MOV M,C	LD (HL),C	A8	XRA B	XOR B
3B	DCX SP	DEC SP	72	MOV M,D	LD (HL),D	A9	XRA C	XOR C
3C	INR A	INC A	73	MOV M,E	LD (HL),E	AA	XRA D	XOR D
3D	DCR A	DEC A	74	MOV M,H	LD (HL),H	AB	XRA E	XOR E
3E	MVI A,dd	LD A,dd	75	MOV M,L	LD (HL),L	AC	XRA H	XOR H
3F	CMC	CCF	76	HLT	HALT	AD	XRA L	XOR L
40	MOV B,B	LD B,B	77	MOV M,A	LD (HL),A	AE	XRA M	XOR (HL)
41	MOV B,C	LD B,C	78	MOV A,B	LD A,B	AF	XRA A	XOR A
42	MOV B,D	LD B,D	79	MOV A,C	LD A,C	B0	ORA B	OR B
43	MOV B,E	LD B,E	7A	MOV A,D	LD A,D	B1	ORA C	OR C
44	MOV B,H	LD B,H	7B	MOV A,E	LD A,E	B2	ORA D	OR D
45	MOV B,L	LD B,L	7C	MOV A,H	LD A,H	B3	ORA E	OR E
46	MOV B,M	LD B,(HL)	7D	MOV A,L	LD A,L	B4	ORA H	OR H
47	MOV B,A	LD B,A	7E	MOV A,M	LD A,(HL)	B5	ORA L	OR L
48	MOV C,B	LD C,B	7F	MOV A,A	LD A,A	B6	ORA M	OR (HL)
49	MOV C,C	LD C,C	80	ADD B	ADD A,B	B7	ORA A	OR A
4A	MOV C,D	LD C,D	81	ADD C	ADD A,C	B8	CMP B	CP B
4B	MOV C,E	LD C,E	82	ADD D	ADD A,D	B9	CMP C	CP C
4C	MOV C,H	LD C,H	83	ADD E	ADD A,E	BA	CMP D	CP D
4D	MOV C,L	LD C,L	84	ADD H	ADD A,H	BB	CMP E	CP E
4E	MOV C,M	LD C,(HL)	85	ADD L	ADD A,L	BC	CMP H	CP H
4F	MOV C,A	LD C,A	86	ADD M	ADD A,(HL)	BD	CMP L	CP L
50	MOV D,B	LD D,B	87	ADD A	ADD A,A	BE	CMP M	CP (HL)
51	MOV D,C	LD D,C	88	ADC B	ADC A,B	BF	CMP A	CP A
52	MOV D,D	LD D,D	89	ADC C	ADC A,C	C0	RNZ	RET NZ
53	MOV D,E	LD D,E	8A	ADC D	ADC A,D	C1	POP B	POP BC
54	MOV D,H	LD D,H	8B	ADC E	ADC A,E	C2	JNZ aaaa	JP NZ,aaaa
55	MOV D,L	LD D,L	8C	ADC H	ADC A,H	C3	JMP aaaa	JP aaaa
56	MOV D,M	LD D,(HL)	8D	ADC L	ADC A,L	C4	CNZ aaaa	CALL NZ,aaaa
57	MOV D,A	LD D,A	8E	ADC M	ADC A,(HL)	C5	PUSH B	PUSH BC
58	MOV E,B	LD E,B	8F	ADC A	ADC A,A	C6	ADI dd	ADD A,dd
59	MOV E,C	LD E,C	90	SUB B	SUB B	C7	RST 0	RST 00H
5A	MOV E,D	LD E,D	91	SUB C	SUB C	C8	RZ	RET Z
5B	MOV E,E	LD E,E	92	SUB D	SUB D	C9	RET	RET
5C	MOV E,H	LD E,H	93	SUB E	SUB E	CA	JZ aaaa	JP Z,aaaa
5D	MOV E,L	LD E,L	94	SUB H	SUB H	CC	CZ aaaa	CALL Z,aaaa
5E	MOV E,M	LD E,(HL)	95	SUB L	SUB L	CD	CALL aaaa	CALL aaaa

Op	8080/8085	Z80
CE	ACI dd	ADC A,dd
CF	RST 1	RST 08H
D0	RNC	RET NC
D1	POP D	POP DE
D2	JNC aaaa	JP NC,aaaa
D3	OUT dd	OUT dd,A
D4	CNC aaaa	CALL NC,aaaa
D5	PUSH D	PUSH DE
D6	SUI dd	SUB dd
D7	RST 2	RST 10H
D8	RC	RET C
DA	JC aaaa	JP C,aaaa
DB	IN dd	IN A,dd
DC	CC aaaa	CALL C,aaaa
DE	SBI dd	SBC A,dd
DF	RST 3	RST 18H

Op	8080/8085	Z80
E0	RPO	RET PO
E1	POP H	POP HL
E2	JPO aaaa	JP PO,aaaa
E3	XTHL	EX (SP),HL
E4	CPO aaaa	CALL PO,aaaa
E5	PUSH H	PUSH HL
E6	ANI dd	AND dd
E7	RST 4	RST 20H
E8	RPE	RET PE
E9	PCHL	JP (HL)
EA	JPE aaaa	JP PE,aaaa
EB	XCHG	EX DE,HL
EC	CPE aaaa	CALL PE,aaaa
EE	XRI dd	XOR dd
EF	RST 5	RST 28H

Op	8080/8085	Z80
F0	RP	RET P
F1	POP PSW	POP AF
F2	JP aaaa	JP P,aaaa
F3	DI	DI
F4	CP aaaa	CALL P,aaaa
F5	PUSH PSW	PUSH AF
F6	ORI dd	OR dd
F7	RST 6	RST 30H
F8	RM	RET M
F9	SPHL	LD SP,HL
FA	JM aaaa	JP M,aaaa
FB	EI	EI
FC	CM aaaa	CALL M,aaaa
FE	CPI dd	CP dd
FF	RST 7	RST 38H

CONDENSED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED ALPHABETICALLY BY 8085/8080 MNEMONIC

8085	Z80	Op
ACI dd	ADC A,dd	CE
ADC A	ADC A,A	8F
ADC B	ADC A,B	88
ADC C	ADC A,C	89
ADC D	ADC A,D	8A
ADC E	ADC A,E	8B
ADC H	ADC A,H	8C
ADC L	ADC A,L	8D
ADC M	ADC A,(HL)	8E
ADD A	ADD A,A	87
ADD B	ADD A,B	80
ADD C	ADD A,C	81
ADD D	ADD A,D	82
ADD E	ADD A,E	83
ADD H	ADD A,H	84
ADD L	ADD A,L	85
ADD M	ADD A,(HL)	86
ADI dd	ADD A,dd	C6
ANA A	AND A	A7
ANA B	AND B	A0
ANA C	AND C	A1
ANA D	AND D	A2
ANA E	AND E	A3
ANA H	AND H	A4
ANA L	AND L	A5
ANA M	AND (HL)	A6
ANI dd	AND dd	E6
CALL aaaa	CALL aaaa	CD
CC aaaa	CALL C,aaaa	DC
CM aaaa	CALL M,aaaa	FC
CMA	CPL	2F
CMC	CCF	3F
CMP A	CP A	BF
CMP B	CP B	B8

8085	Z80	Op
CMP C	CP C	B9
CMP D	CP D	BA
CMP E	CP E	BB
CMP H	CP H	BC
CMP L	CP L	BD
CMP M	CP (HL)	BE
CNC aaaa	CALL NC,aaaa	D4
CNZ aaaa	CALL NZ,aaaa	C4
CP aaaa	CALL P,aaaa	F4
CPE aaaa	CALL PE,aaaa	EC
CPI dd	CP dd	FE
CPO aaaa	CALL PO,aaaa	E4
CZ aaaa	CALL Z,aaaa	CC
DAA	DAA	27
DAD B	ADD HL,BC	09
DAD D	ADD HL,DE	19
DAD H	ADD HL,HL	29
DAD SP	ADD HL,SP	39
DCR A	DEC A	3D
DCR B	DEC B	05
DCR C	DEC C	0D
DCR D	DEC D	15
DCR E	DEC E	1D
DCR H	DEC H	25
DCR L	DEC L	2D
DCR M	DEC (HL)	35
DCX B	DEC BC	0B
DCX D	DEC DE	1B
DCX H	DEC HL	2B
DCX SP	DEC SP	3B
DI	DI	F3
EI	EI	FB
HLT	HALT	76
IN dd	IN A,dd	DB

8085	Z80	Op
INR A	INC A	3C
INR B	INC B	04
INR C	INC C	0C
INR D	INC D	14
INR E	INC E	1C
INR H	INC H	24
INR L	INC L	2C
INR M	INC (HL)	34
INX B	INC BC	03
INX D	INC DE	13
INX H	INC HL	23
INX SP	INC SP	33
JC aaaa	JP C,aaaa	DA
JM aaaa	JP M,aaaa	FA
JMP aaaa	JP aaaa	C3
JNC aaaa	JP NC,aaaa	D2
JNZ aaaa	JP NZ,aaaa	C2
JP aaaa	JP P,aaaa	F2
JPE aaaa	JP PE,aaaa	EA
JPO aaaa	JP PO,aaaa	E2
JZ aaaa	JP Z,aaaa	CA
LDA aaaa	LD A,(aaaa)	3A
LDAX B	LD A,(BC)	0A
LDAX D	LD A,(DE)	1A
LHLD aaaa	LD HL,(aaaa)	2A
LXI B,dddd	LD BC,dddd	01
LXI D,dddd	LD DE,dddd	11
LXI H,dddd	LD HL,dddd	21
LXI SP,dddd	LD SP,dddd	31
MOV A,A	LD A,A	7F
MOV A,B	LD A,B	78
MOV A,C	LD A,C	79
MOV A,D	LD A,D	7A
MOV A,E	LD A,E	7B

CONDENSED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED ALPHABETICALLY BY 8085/8080 MNEMONIC (*Continued*)

8085	Z80	Op	8085	Z80	Op	8085	Z80	Op
MOV A,H	LD A,H	7C	MOV L,H	LD L,H	6C	RPE	RET PE	E8
MOV A,L	LD A,L	7D	MOV L,L	LD L,L	6D	RPO	RET PO	E0
MOV A,M	LD A,(HL)	7E	MOV L,M	LD L,(HL)	6E	RRC	RRCA	0F
MOV B,A	LD B,A	47	MOV M,A	LD (HL),A	77	RST 0	RST 00H	C7
MOV B,B	LD B,B	40	MOV M,B	LD (HL),B	70	RST 1	RST 08H	CF
MOV B,C	LD B,C	41	MOV M,C	LD (HL),C	71	RST 2	RST 10H	D7
MOV B,D	LD B,D	42	MOV M,D	LD (HL),D	72	RST 3	RST 18H	DF
MOV B,E	LD B,E	43	MOV M,E	LD (HL),E	73	RST 4	RST 20H	E7
MOV B,H	LD B,H	44	MOV M,H	LD (HL),H	74	RST 5	RST 28H	EF
MOV B,L	LD B,L	45	MOV M,L	LD (HL),L	75	RST 6	RST 30H	F7
MOV B,M	LD B,(HL)	46	MVI A,dd	LD A,dd	3E	RST 7	RST 38H	FF
MOV C,A	LD C,A	4F	MVI B,dd	LD B,dd	06	RZ	RET Z	C8
MOV C,B	LD C,B	48	MVI C,dd	LD C,dd	0E	SBB A	SBC A,A	9F
MOV C,C	LD C,C	49	MVI D,dd	LD D,dd	16	SBB B	SBC A,B	98
MOV C,D	LD C,D	4A	MVI E,dd	LD E,dd	1E	SBB C	SBC A,C	99
MOV C,E	LD C,E	4B	MVI H,dd	LD H,dd	26	SBB D	SBC A,D	9A
MOV C,H	LD C,H	4C	MVI L,dd	LD L,dd	2E	SBB E	SBC A,E	9B
MOV C,L	LD C,L	4D	MVI M,dd	LD (HL),dd	36	SBB H	SBC A,H	9C
MOV C,M	LD C,(HL)	4E	NOP	NOP	00	SBB L	SBC A,L	9D
MOV D,A	LD D,A	57	ORA A	OR A	B7	SBB M	SBC A,(HL)	9E
MOV D,B	LD D,B	50	ORA B	OR B	B0	SBI dd	SBC A,dd	DE
MOV D,C	LD D,C	51	ORA C	OR C	B1	SHLD aaaa	LD (aaaa),HL	22
MOV D,D	LD D,D	52	ORA D	OR D	B2	SPHL	LD SP,HL	F9
MOV D,E	LD D,E	53	ORA E	OR E	B3	STA aaaa	LD (aaaa),A	32
MOV D,H	LD D,H	54	ORA H	OR H	B4	STAX B	LD (BC),A	02
MOV D,L	LD D,L	55	ORA L	OR L	B5	STAX D	LD (DE),A	12
MOV D,M	LD D,(HL)	56	ORA M	OR (HL)	B6	STC	SCF	37
MOV E,A	LD E,A	5F	ORI dd	OR dd	F6	SUB A	SUB A	97
MOV E,B	LD E,B	58	OUT dd	OUT dd,A	D3	SUB B	SUB B	90
MOV E,C	LD E,C	59	PCHL	JP (HL)	E9	SUB C	SUB C	91
MOV E,D	LD E,D	5A	POP B	POP BC	C1	SUB D	SUB D	92
MOV E,E	LD E,E	5B	POP D	POP DE	D1	SUB E	SUB E	93
MOV E,H	LD E,H	5C	POP H	POP HL	E1	SUB H	SUB H	94
MOV E,L	LD E,L	5D	POP PSW	POP AF	F1	SUB L	SUB L	95
MOV E,M	LD E,(HL)	5E	PUSH B	PUSH BC	C5	SUB M	SUB (HL)	96
MOV H,A	LD H,A	67	PUSH D	PUSH DE	D5	SUI dd	SUB dd	D6
MOV H,B	LD H,B	60	PUSH H	PUSH HL	E5	XCHG	EX DE,HL	EB
MOV H,C	LD H,C	61	PUSH PSW	PUSH AF	F5	XRA A	XOR A	AF
MOV H,D	LD H,D	62	RAL	RLA	17	XRA B	XOR B	A8
MOV H,E	LD H,E	63	RAR	RRA	1F	XRA C	XOR C	A9
MOV H,H	LD H,H	64	RC	RET C	D8	XRA D	XOR D	AA
MOV H,L	LD H,L	65	RET	RET	C9	XRA E	XOR E	AB
MOV H,M	LD H,(HL)	66	RLC	RLCA	07	XRA H	XOR H	AC
MOV L,A	LD L,A	6F	RM	RET M	F8	XRA L	XOR L	AD
MOV L,B	LD L,B	68	RNC	RET NC	D0	XRA M	XOR (HL)	AE
MOV L,C	LD L,C	69	RNZ	RET NZ	C0	XRI dd	XOR dd	EE
MOV L,D	LD L,D	6A	RP	RET P	F0	XTHL	EX (SP),HL	E3
MOV L,E	LD L,E	6B						

CONDENSED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED ALPHABETICALLY BY Z80 MNEMONIC

Z80	8080/8085	Op
ADC A,(HL)	ADC M	8E
ADC A,A	ADC A	8F
ADC A,B	ADC B	88
ADC A,C	ADC C	89
ADC A,D	ADC D	8A
ADC A,dd	ACI dd	CE
ADC A,E	ADC E	8B
ADC A,H	ADC H	8C
ADC A,L	ADC L	8D
ADD A,(HL)	ADD M	86
ADD A,A	ADD A	87
ADD A,B	ADD B	80
ADD A,C	ADD C	81
ADD A,D	ADD D	82
ADD A,dd	ADI dd	C6
ADD A,E	ADD E	83
ADD A,H	ADD H	84
ADD A,L	ADD L	85
ADD HL,BC	DAD B	09
ADD HL,DE	DAD D	19
ADD HL,HL	DAD H	29
ADD HL,SP	DAD SP	39
AND (HL)	ANA M	A6
AND A	ANA A	A7
AND B	ANA B	A0
AND C	ANA C	A1
AND D	ANA D	A2
AND dd	ANI dd	E6
AND E	ANA E	A3
AND H	ANA H	A4
AND L	ANA L	A5
CALL aaaa	CALL aaaa	CD
CALL C,aaaa	CC aaaa	DC
CALL M,aaaa	CM aaaa	FC
CALL NC,aaaa	CNC aaaa	D4
CALL NZ,aaaa	CNZ aaaa	C4
CALL P,aaaa	CP aaaa	F4
CALL PE,aaaa	CPE aaaa	EC
CALL PO,aaaa	CPO aaaa	E4
CALL Z,aaaa	CZ aaaa	CC
CCF	CMC	3F
CP (HL)	CMP M	BE
CP A	CMP A	BF
CP B	CMP B	B8
CP C	CMP C	B9
CP D	CMP D	BA
CP dd	CPI dd	FE
CP E	CMP E	BB
CP H	CMP H	BC
CP L	CMP L	BD
CPL	CMA	2F
DAA	DAA	27
DEC (HL)	DCR M	35
DEC A	DCR A	3D
DEC B	DCR B	05
DEC BC	DCX B	0B
DEC C	DCR C	0D
DEC D	DCR D	15
DEC DE	DCX D	1B
DEC E	DCR E	1D
DEC H	DCR H	25
DEC HL	DCX H	2B
DEC L	DCR L	2D
DEC SP	DCX SP	3B
DI	DI	F3
EI	EI	FB
EX (SP),HL	XTHL	E3
EX DE,HL	XCHG	EB
HALT	HLT	76
IN A,dd	IN dd	DB
INC (HL)	INR M	34
INC A	INR A	3C
INC B	INR B	04
INC BC	INX B	03
INC C	INR C	0C
INC D	INR D	14
INC DE	INX D	13
INC E	INR E	1C
INC H	INR H	24
INC HL	INX H	23
INC L	INR L	2C
INC SP	INX SP	33
JP (HL)	PCHL	E9
JP aaaa	JMP aaaa	C3
JP C,aaaa	JC aaaa	DA
JP M,aaaa	JM aaaa	FA
JP NC,aaaa	JNC aaaa	D2
JP NZ,aaaa	JNZ aaaa	C2
JP P,aaaa	JP aaaa	F2
JP PE,aaaa	JPE aaaa	EA
JP PO,aaaa	JPO aaaa	E2
JP Z,aaaa	JZ aaaa	CA
LD (aaaa),A	STA aaaa	32
LD (aaaa),HL	SHLD aaaa	22
LD (BC),A	STAX B	02
LD (DE),A	STAX D	12
LD (HL),A	MOV M,A	77
LD (HL),B	MOV M,B	70
LD (HL),C	MOV M,C	71
LD (HL),D	MOV M,D	72
LD (HL),dd	MVI M,dd	36
LD (HL),E	MOV M,E	73
LD (HL),H	MOV M,H	74
LD (HL),L	MOV M,L	75
LD A,(aaaa)	LDA aaaa	3A
LD A,(BC)	LDAX B	0A
LD A,(DE)	LDAX D	1A
LD A,(HL)	MOV A,M	7E
LD A,A	MOV A,A	7F
LD A,B	MOV A,B	78
LD A,C	MOV A,C	79
LD A,D	MOV A,D	7A
LD A,dd	MVI A,dd	3E
LD A,E	MOV A,E	7B
LD A,H	MOV A,H	7C
LD A,L	MOV A,L	7D
LD B,(HL)	MOV B,M	46
LD B,A	MOV B,A	47
LD B,B	MOV B,B	40
LD B,C	MOV B,C	41
LD BC,dddd	LXI B,dddd	01
LD B,D	MOV B,D	42
LD B,dd	MVI B,dd	06
LD B,E	MOV B,E	43
LD B,H	MOV B,H	44
LD B,L	MOV B,L	45
LD C,(HL)	MOV C,M	4E
LD C,A	MOV C,A	4F
LD C,B	MOV C,B	48
LD C,C	MOV C,C	49
LD C,D	MOV C,D	4A
LD C,dd	MVI C,dd	0E
LD C,E	MOV C,E	4B
LD C,H	MOV C,H	4C
LD C,L	MOV C,L	4D
LD D,(HL)	MOV D,M	56
LD D,A	MOV D,A	57
LD D,B	MOV D,B	50
LD D,C	MOV D,C	51
LD D,D	MOV D,D	52
LD D,dd	MVI D,dd	16
LD D,E	MOV D,E	53
LD DE,dddd	LXI D,dddd	11
LD D,H	MOV D,H	54
LD D,L	MOV D,L	55
LD E,(HL)	MOV E,M	5E
LD E,A	MOV E,A	5F
LD E,B	MOV E,B	58
LD E,C	MOV E,C	59
LD E,D	MOV E,D	5A
LD E,dd	MVI E,dd	1E
LD E,E	MOV E,E	5B
LD E,H	MOV E,H	5C
LD E,L	MOV E,L	5D
LD H,(HL)	MOV H,M	66
LD H,A	MOV H,A	67
LD H,B	MOV H,B	60
LD H,C	MOV H,C	61
LD H,D	MOV H,D	62
LD H,dd	MVI H,dd	26
LD H,E	MOV H,E	63
LD H,H	MOV H,H	64
LD H,L	MOV H,L	65
LD HL,(aaaa)	LHLD aaaa	2A
LD HL,dddd	LXI H,dddd	21

CONDENSED TABLE OF 8085/8080 AND Z80 (8080 SUBSET) INSTRUCTIONS LISTED ALPHABETICALLY BY Z80 MNEMONIC *(Continued)*

Z80	8080/8085	Op	Z80	8080/8085	Op	Z80	8080/8085	Op
LD L,(HL)	MOV L,M	6E	PUSH BC	PUSH B	C5	SBC A,B	SBB B	98
LD L,A	MOV L,A	6F	PUSH DE	PUSH D	D5	SBC A,C	SBB C	99
LD L,B	MOV L,B	68	PUSH HL	PUSH H	E5	SBC A,D	SBB D	9A
LD L,C	MOV L,C	69	RET	RET	C9	SBC A,dd	SBI dd	DE
LD L,D	MOV L,D	6A	RET C	RC	D8	SBC A,E	SBB E	9B
LD L,dd	MVI L,dd	2E	RET M	RM	F8	SBC A,H	SBB H	9C
LD L,E	MOV L,E	6B	RET NC	RNC	D0	SBC A,L	SBB L	9D
LD L,H	MOV L,H	6C	RET NZ	RNZ	C0	SCF	STC	37
LD L,L	MOV L,L	6D	RET P	RP	F0	SUB (HL)	SUB M	96
LD SP,dddd	LXI SP,dddd	31	RET PE	RPE	E8	SUB A	SUB A	97
LD SP,HL	SPHL	F9	RET PO	RPO	E0	SUB dd	SUI dd	D6
NOP	NOP	00	RET Z	RZ	C8	SUB B	SUB B	90
OR (HL)	ORA M	B6	RLA	RAL	17	SUB C	SUB C	91
OR A	ORA A	B7	RLCA	RLC	07	SUB D	SUB D	92
OR B	ORA B	B0	RRA	RAR	1F	SUB E	SUB E	93
OR C	ORA C	B1	RRCA	RRC	0F	SUB H	SUB H	94
OR D	ORA D	B2	RST 00H	RST 0	C7	SUB L	SUB L	95
OR dd	ORI dd	F6	RST 08H	RST 1	CF	XOR (HL)	XRA M	AE
OR E	ORA E	B3	RST 10H	RST 2	D7	XOR A	XRA A	AF
OR H	ORA H	B4	RST 18H	RST 3	DF	XOR B	XRA B	A8
OR L	ORA L	B5	RST 20H	RST 4	E7	XOR C	XRA C	A9
OUT dd,A	OUT dd	D3	RST 28H	RST 5	EF	XOR D	XRA D	AA
POP AF	POP PSW	F1	RST 30H	RST 6	F7	XOR dd	XRI dd	EE
POP BC	POP B	C1	RST 38H	RST 7	FF	XOR E	XRA E	AB
POP DE	POP D	D1	SBC A,(HL)	SBB M	9E	XOR H	XRA H	AC
POP HL	POP H	E1	SBC A,A	SBB A	9F	XOR L	XRA L	AD
PUSH AF	PUSH PSW	F5						

EXPANDED TABLE OF 6800 INSTRUCTIONS LISTED BY CATEGORY

Mne-monic	Operation	Boolean/Arith. Operation	Flags HINZVC	Address Mode	Assembler Notation	Op	~	#	Notes
				CPU Control Instructions					
NOP	No OPeration	Nothing	xxxxxx	Implied	NOP	01	2	1	Only the program counter is incremented. No operation occurs.
WAI	WAIt for interrupt	PC + 1 → PC PC_L → S PC_H → S X_L → S X_H → S A → S B → S CCR → S	xIxxxx	Implied	WAI	3E	9	1	After those actions shown in the "Boolean/Arithmetic Operation" column take place, the current program is suspended. If I=0 and the Interrupt Request line is taken low then I=1 and the microprocessor will begin to execute a program whose address is found in memory locations FFF8 and FFF9.

Data Transfer Instructions

Mnemonic	Operation	Boolean/Arith. Operation	Flags HINZVC	Address Mode	Assembler Notation	Op	~	#	Notes
LDAA	LoaD Accumulator A	$M \rightarrow A$	xxNZ0x	Immediate	LDAA #$dd	86	2	2	
				Direct	LDAA $aa	96	3	2	
				Indexed	LDAA $ff,X	A6	5	2	
				Extended	LDAA $aaaa	B6	4	3	
LDAB	LoaD Accumulator B	$M \rightarrow B$	xxNZ0x	Immediate	LDAB #$dd	C6	2	2	
				Direct	LDAB $aa	D6	2	2	
				Indexed	LDAB $ff,X	E6	5	2	
				Extended	LDAB $aaaa	F6	4	3	
STAA	STore Accumulator A	$A \rightarrow M$	xxNZ0x	Direct	STAA $aa	97	4	2	
				Indexed	STAA $ff,X	A7	6	2	
				Extended	STAA $aaaa	B7	5	3	
STAB	STore Accumulator B	$B \rightarrow M$	xxNZ0x	Direct	STAB $aa	D7	4	2	
				Indexed	STAB $ff,X	E7	6	2	
				Extended	STAB $aaaa	F7	5	3	
TAB	Transfer A to B	$A \rightarrow B$	xxNZ0x	Implied	TAB	16	2	1	
TBA	Transfer B to A	$B \rightarrow A$	xxNZ0x	Implied	TBA	17	2	1	
LDX	LoaD X register	$M \rightarrow X_H$ $(M + 1) \rightarrow X_L$	xxNZ0x	Immediate	LDX #$dddd	CE	3	3	
				Direct	LDX $aa	DE	4	2	
				Indexed	LDX $ff,X	EE	6	2	
				Extended	LDX $aaaa	FE	5	3	
STX	STore X register	$X_H \rightarrow M$ $X_L \rightarrow (M + 1)$	xxNZ0x	Direct	STX $aa	DF	5	2	
				Indexed	STX $ff,X	EF	7	2	
				Extended	STX $aaaa	FF	6	3	
CLR	CLeaR memory location	$00 \rightarrow M$	xx0100	Indexed	CLR $ff,X	6F	7	2	
				Extended	CLR $aaaa	7F	6	3	
CLRA	CLeaR accumulator A	$00 \rightarrow A$	xx0100	Implied	CLRA	4F	2	1	
CLRB	CLeaR accumulator B	$00 \rightarrow B$	xx0100	Implied	CLRB	5F	2	1	

Flag Instructions

Mnemonic	Operation	Boolean/Arith. Operation	Flags HINZVC	Address Mode	Assembler Notation	Op	~	#	Notes
CLC	CLear Carry flag	$0 \rightarrow C$	xxxxx0	Implied	CLC	0C	2	1	

EXPANDED TABLE OF 6800 INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Mne-monic	Operation	Boolean/Arith. Operation	Flags HINZVC	Address Mode	Assembler Notation	Op	~	#	Notes
CLI	CLear Interrupt flag	0 → I	x0xxxx	Implied	CLI	0E	2	1	
CLV	CLear oVerflow flag	0 → V	xxxxVx	Implied	CLV	0A	2	1	
SEC	SEt Carry flag	1 → C	xxxxx1	Implied	SEC	0D	2	1	
SEI	SEt Interrupt flag	1 → I	x1xxxx	Implied	SEI	0F	2	1	
SEV	SEt oVerflow flag	1 → V	xxxx1x	Implied	SEV	0B	2	1	
TAP	Transfer Accumulator A to Processor condition code register	A → CCR	HINZVC	Implied	TAP	06	2	1	
TPA	Transfer Processor condition code register to accumulator A	CCR → A	xxxxxx	Implied	TPA	07	2	1	

Arithmetic Instructions

Mne-monic	Operation	Boolean/Arith. Operation	Flags HINZVC	Address Mode	Assembler Notation	Op	~	#	Notes
ADDA	ADD accumulator A to memory location	A + M → A	HxNZVC	Immediate	ADDA #$dd	8B	2	2	
				Direct	ADDA $aa	9B	3	2	
				Indexed	ADDA $ff,X	AB	5	2	
				Extended	ADDA $aaaa	BB	4	3	
ADDB	ADD accumulator B to memory location	B + M → B	HxNZVC	Immediate	ADDB #$dd	CB	2	2	
				Direct	ADDB $aa	DB	3	2	
				Indexed	ADDB $ff,X	EB	5	2	
				Extended	ADDB $aaaa	FB	4	3	
ABA	Add accumulator B to accumulator A	A + B → A	HxNZVC	Implied	ABA	1B	2	1	
ADCA	AdD with Carry accumulator A to memory location	A + M + C → A	HxNZVC	Immediate	ADCA #$dd	89	2	2	
				Direct	ADCA $aa	99	3	2	
				Indexed	ADCA $ff,X	A9	5	2	
				Extended	ADCA $aaaa	B9	4	3	
ADCB	AdD with Carry accumulator B to memory location	B + M + C → B	HxNZVC	Immediate	ADCB #$dd	C9	2	2	
				Direct	ADCB $aa	D9	3	2	
				Indexed	ADCB $ff,X	E9	5	2	
				Extended	ADCB $aaaa	F9	4	3	

Mne-monic	Operation	Boolean/Arith. Operation	Flags HINZVC	Address Mode	Assembler Notation	Op	~	#	Notes
SUBA	SUBtract memory location from accumulator **A**	A - M → A	xxNZVC	Immediate	SUBA #$dd	80	2	2	
				Direct	SUBA $aa	90	3	2	
				Indexed	SUBA $ff,X	A0	5	2	
				Extended	SUBA $aaaa	B0	4	3	
SUBB	SUBtract memory location from accumulator **B**	B - M → B	xxNZVC	Immediate	SUBB #$dd	C0	2	2	
				Direct	SUBB $aa	D0	3	2	
				Indexed	SUBB $ff,X	E0	5	2	
				Extended	SUBB $aaaa	F0	4	3	
SBA	Subtract accumulator **B** from accumulator **A**	A - B → A	xxNZVC	Implied	SBA	10	2	1	
SBCA	SuBtract with Carry memory location from accumulator **A**	A - M - C → A	xxNZVC	Immediate	SBCA #$dd	82	2	2	
				Direct	SBCA $aa	92	3	2	
				Indexed	SBCA $ff,X	A2	5	2	
				Extended	SBCA $aaaa	B2	4	3	
SBCB	SuBtract with Carry memory location from accumulator **B**	B - M - C → B	xxNZVC	Immediate	SBCB #$dd	C2	2	2	
				Direct	SBCB $aa	D2	3	2	
				Indexed	SBCB $ff,X	E2	5	2	
				Extended	SBCB $aaaa	F2	4	3	
DAA	Decimal Adjust accumulator **A**	(converts binary number into BCD number)	xxNZVC	Implied	DAA	19	2	1	Converts the number in A to the BCD number it would be if the last two operands had been BCD numbers.

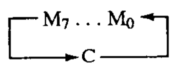
Logical Instructions

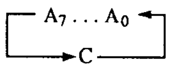
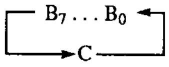
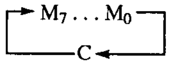
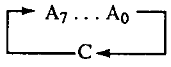
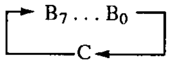
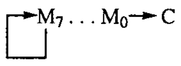
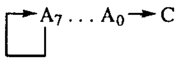
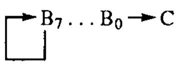
ANDA	**AND** accumulator **A** with memory location	A AND M → A	xxNZ0x	Immediate	ANDA #$dd	84	2	2	
				Direct	ANDA $aa	94	3	2	
				Indexed	ANDA $ff,X	A4	5	2	
				Extended	ANDA $aaaa	B4	4	3	
ANDB	**AND** accumulator **B** with memory location	B AND M → B	xxNZ0x	Immediate	ANDB #$dd	C4	2	2	
				Direct	ANDB $aa	D4	3	2	
				Indexed	ANDB $ff,X	E4	5	2	
				Extended	ANDB $aaaa	F4	4	3	
ORAA	**OR** Accumulator **A** with memory location	A OR M → A	xxNZ0x	Immediate	ORAA #$dd	8A	2	2	
				Direct	ORAA $aa	9A	3	2	
				Indexed	ORAA $ff,X	AA	5	2	
				Extended	ORAA $aaaa	BA	4	3	
ORAB	**OR** Accumulator **B** with memory location	B OR M → B	xxNZ0x	Immediate	ORAB #$dd	CA	2	2	
				Direct	ORAB $aa	DA	3	2	
				Indexed	ORAB $ff,X	EA	5	2	
				Extended	ORAB $aaaa	FA	4	3	

EXPANDED TABLE OF 6800 INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Mne-monic	Operation	Boolean/Arith. Operation	Flags HINZVC	Address Mode	Assembler Notation	Op	~	#	Notes
EORA	Exclusively **OR** accumulator **A** with memory location	A EOR M $\rightarrow$ A	xxNZ0x	Immediate	EORA #$dd	88	2	2	
				Direct	EORA $aa	98	3	2	
				Indexed	EORA $ff,X	A8	5	2	
				Extended	EORA $aaaa	B8	4	3	
EORB	Exclusively **OR** accumulator **A** with memory location	B EOR M $\rightarrow$ B	xxNZ0x	Immediate	EORB #$dd	C8	2	2	
				Direct	EORB $aa	D8	3	2	
				Indexed	EORB $ff,X	E8	5	2	
				Extended	EORB $aaaa	F8	4	3	
BITA	**BIT** test accumulator **A**	A AND M	xxNZ0x	Immediate	BITA #$dd	85	2	2	Accumulator A and a memory location are ANDed but neither is changed. However, flags N and Z are affected accordingly.
				Direct	BITA $aa	95	3	2	
				Indexed	BITA $ff,X	A5	5	2	
				Extended	BITA $aaaa	B5	4	3	
BITB	**BIT** test accumulator **B**	B AND M	xxNZ0x	Immediate	BITB #$dd	C5	2	2	Accumulator B and a memory location are ANDed but neither is changed. However, flags N and Z are affected accordingly.
				Direct	BITB $aa	D5	3	2	
				Indexed	BITB $ff,X	E5	5	2	
				Extended	BITB $aaaa	F5	4	3	
COM	**COM**plement memory location (1's complement)	$\overline{M} \rightarrow M$	xxNZ01	Indexed	COM $ff,X	63	7	2	
				Extended	COM $aaaa	73	6	2	
COMA	**COM**plement accumulator **A** (1's complement)	$\overline{A} \rightarrow A$	xxNZ01	Implied	COMA	43	2	1	
COMB	**COM**plement accumulator **B** (1's complement)	$\overline{B} \rightarrow B$	xxNZ01	Implied	COMB	53	2	1	
NEG	**NEG**ate memory location (2's complement)	00 - M $\rightarrow$ M	xxNZVC	Indexed	NEG $ff,X	60	7	2	Affects the carry flag as if the memory location had been subtracted from zero.
				Extended	NEG $aaaa	70	6	3	
NEGA	**NEG**ate accumulator **A** (2's complement)	00 - A $\rightarrow$ A	xxNZVC	Implied	NEGA	40	2	1	Affects the carry flag as if accumulator A had been subtracted from zero.
NEGB	**NEG**ate accumulator **B** (2's complement)	00 - B $\rightarrow$ B	xxNZVC	Implied	NEGB	50	2	1	Affects the carry flag as if accumulator B had been subtracted from zero.

Rotate and Shift Instructions

Mne-monic	Operation	Boolean/Arith. Operation	Flags HINZVC	Address Mode	Assembler Notation	Op	~	#	Notes
ROL	**RO**tate memory location **L**eft		xxNZVC	Indexed	ROL $ff,X	69	7	2	
				Extended	ROL $aaaa	79	6	3	

Mne-monic	Operation	Boolean/Arith. Operation	Flags HINZVC	Address Mode	Assembler Notation	Op	~	#	Notes
ROLA	ROtate to the Left accumulator A		xxNZVC	Implied	ROLA	49	2	1	
ROLB	ROtate to the Left accumulator B		xxNZVC	Implied	ROLB	59	2	1	
ROR	ROtate memory location Right		xxNZVC	Indexed Extended	ROR $ff,X ROR $aaaa	66 76	7 6	2 3	
RORA	ROtate to the Right accumulator A		xxNZVC	Implied	RORA	46	2	1	
RORB	ROtate to the Right accumulator B		xxNZVC	Implied	RORB	56	2	1	
ASL	Arithmetic Shift Left memory location	$C \leftarrow M_7 \ldots M_0 \leftarrow 0$	xxNZVC	Indexed Extended	ASL $ff,X ASL $aaaa	68 78	7 6	2 3	
ASLA	Arithmetic Shift Left accumulator A	$C \leftarrow A_7 \ldots A_0 \leftarrow 0$	xxNZVC	Implied	ASLA	48	2	1	
ASLB	Arithmetic Shift Left accumulator B	$C \leftarrow B_7 \ldots B_0 \leftarrow 0$	xxNZVC	Implied	ASLB	58	2	1	
ASR	Arithmetic Shift Right memory location		xxNZVC	Indexed Extended	ASR $ff,X ASR $aaaa	67 77	7 6	2 3	
ASRA	Arithmetic Shift Right accumulator A		xxNZVC	Implied	ASRA	47	2	1	
ASRB	Arithmetic Shift Right accumulator B		xxNZVC	Implied	ASRB	57	2	1	
LSR	Logical Shift Right memory location	$0 \rightarrow M_7 \ldots M_0 \rightarrow C$	xx0ZVC	Indexed Extended	LSR $ff,X LSR $aaaa	64 74	7 6	2 3	
LSRA	Logical Shift Right accumulator A	$0 \rightarrow A_7 \ldots A_0 \rightarrow C$	xx0ZVC	Implied	LSRA	44	2	1	
LSRB	Logical Shift Right accumulator B	$0 \rightarrow B_7 \ldots B_0 \rightarrow C$	xx0ZVC	Implied	LSRB	54	2	1	

EXPANDED TABLE OF 6800 INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Mne-monic	Operation	Boolean/Arith. Operation	Flags HINZVC	Address Mode	Assembler Notation	Op	~	#	Notes
			Increment and Decrement Instructions						
INC	INCrement memory location	M + 1 → M	xxNZVx	Indexed	INC $ff,X	6C	7	2	
				Extended	INC $aaaa	7C	6	3	
INCA	INCrement accumulator A	A + 1 → A	xxNZVx	Implied	INCA	4C	2	1	
INCB	INCrement accumulator B	B + 1 → B	xxNZVx	Implied	INCB	5C	2	1	
DEC	DECrement memory location	M - 1 → M	xxNZVx	Indexed	DEC $ff,X	6A	7	2	
				Extended	DEC $aaaa	7A	6	3	
DECA	DECrement accumulator A	A - 1 → A	xxNZVx	Implied	DECA	4A	2	1	
DECB	DECrement accumulator B	B - 1 → B	xxNZVx	Implied	DECB	5A	2	1	
INX	INcrement X (index) register	X + 1 → X	xxxZxx	Implied	INX	08	4	1	
DEX	DEcrement X (index) register	X - 1 → X	XXXzXX	Implied	DEX	09	4	1	
			Unconditional Jump Instructions						
JMP	JuMP to memory location	X + ff → PC (indexed) aaaa → PC (extended)	xxxxxx	Indexed	JMP $ff,X	6E	4	2	
				Extended	JMP $aaaa	7E	3	3	
BRA	BRanch Always to memory location	PC + 2 + rr → PC	xxxxxx	Relative	BRA $rr	20	4	2	
			Test (Compare) Instructions						
CMPA	CoMPare memory location to accumulator A	A - M	xxNZVC	Immediate	CMPA #$dd	81	2	2	
				Direct	CMPA $aa	91	3	2	
				Indexed	CMPA $ff,X	A1	5	2	
				Extended	CMPA $aaaa	B1	4	3	
CMPB	CoMPare memory location to accumulator B	B - M	xxNZVC	Immediate	CMPB #$dd	C1	2	2	
				Direct	CMPB $aa	D1	3	2	
				Indexed	CMPB $ff,X	E1	5	2	
				Extended	CMPB $aaaa	F1	4	3	

Mne-monic	Operation	Boolean/Arith. Operation	Flags HINZVC	Address Mode	Assembler Notation	Op	~	#	Notes
CBA	Compare accumulator B to accumulator A	A - B	xxNZVC	Implied	CBA	11	2	1	
CPX	ComPare memory location to X (index) register	X_H - M X_L - (M+1)	xxNZVx	Immediate Direct Indexed Extended	CPX #$dddd CPX $aa CPX $ff,X CPX $aaaa	8C 9C AC BC	3 4 6 5	3 2 2 3	
TST	TEsT memory location for zero or minus	M - 00	xxNZ00	Indexed Extended	TST $ff,X TST $aaaa	6D 7D	7 6	2 3	
TSTA	TEsT accumulator A for zero or minus	A - 00	xxNZ00	Implied	TSTA	4D	2	1	
TSTB	TEsT accumulator B for zero or minus	B - 00	xxNZ00	Implied	TSTB	5D	2	1	

Conditional Jump (Branch) Instructions

BCC	Branch if Carry Clear	PC + 2 + rr → PC if C=0	xxxxxx	Relative	BCC $rr	24	4	2	
BCS	Branch if Carry Set	PC + 2 + rr → PC if C=1	xxxxxx	Relative	BCS $rr	25	4	2	
BEQ	Branch if result of last operation was EQual to zero	PC + 2 + rr → PC if Z=1	xxxxxx	Relative	BEQ $rr	27	4	2	
BGE	Branch if Greater than or Equal to zero	PC + 2 + rr → PC if N EOR V = 0	xxxxxx	Relative	BGE $rr	2C	4	2	This branch occurs after the instructions CBA, CMP, SBA, or SUB if the 2's-complement minuend is greater than or equal to the 2's-complement subtrahend creating an answer which is greater than or equal to zero.
BGT	Branch if Greater Than zero	PC + 2 + rr → PC if Z AND (N EOR V) = 0	xxxxxx	Relative	BGT $rr	2E	4	2	This branch occurs after the instructions CBA, CMP, SBA, or SUB if the 2's-complement minuend is greater than the 2's-complement subtrahend, creating an answer which is greater than zero.

EXPANDED TABLE OF 6800 INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Mne-monic	Operation	Boolean/Arith. Operation	Flags HINZVC	Address Mode	Assembler Notation	Op	~	#	Notes
BHI	Branch if HIgher	PC + 2 + rr → PC if C AND Z = 0	xxxxxx	Relative	BHI $rr	22	4	2	This branch occurs after the instructions CBA, CMP, SBA, or SUB if the unsigned binary minuend is greater than the unsigned binary subtrahend.
BLE	Branch if Less than or Equal to zero	PC + 2 + rr → PC if Z AND (N EOR V) = 1	xxxxxx	Relative	BLE $rr	2F	4	2	This branch occurs after the instructions CBA, CMP, SBA, or SUB if the 2's-complement minuend is less than or equal to the 2's-complement subtrahend, creating an answer which is less than or equal to zero.
BLS	Branch if Lower or the Same	PC + 2 + rr → PC if C OR Z = 1	xxxxxx	Relative	BLS $rr	23	4	2	This branch occurs after the instructions CBA, CMP, SBA, or SUB if the unsigned binary minuend is less than or equal to the unsigned binary subtrahend.
BLT	Branch if Less Than zero	PC + 2 + rr → PC if N EOR V = 1	xxxxxx	Relative	BLT $rr	2D	4	2	This branch occurs after the instructions CBA, CMP, SBA, or SUB if the 2's-complement minuend is less than the 2's-complement subtrahend, creating an answer which is less than zero.
BMI	Branch is MInus	PC + 2 + rr → PC if N=1	xxxxxx	Relative	BMI $rr	2B	4	2	
BNE	Branch if Not Equal to zero	PC + 2 + rr → PC if Z=1	xxxxxx	Relative	BNE $rr	26	4	2	
BVC	Branch if oVerflow Clear	PC + 2 + rr → PC if V=0	xxxxxx	Relative	BVC $rr	28	4	2	
BVS	Branch if oVerflow Set	PC + 2 + rr → PC if V=1	xxxxxx	Relative	BVS $rr	29	4	2	
BPL	Branch if PLus	PC + 2 + rr → PC if N=0	xxxxxx	Relative	BPL $rr	2A	4	2	

Mnemonic	Operation	Boolean/Arith. Operation	Flags HINZVC	Address Mode	Assembler Notation	Op	~	#	Notes
		Subroutine Instructions							
JSR	Jump SubRoutine	PC + 2 → PC PC_L → S PC_H → S SP - 2 → SP (ff+X) → PC	xxxxxx	Indexed	JSR $ff,X	AD	8	2	The program counter is incremented by 2 (Indexed) or 3 (Extended) and the program counter is pushed onto the stack 1 byte at a time. At the memory location indicated by the addressing mode will be found the address of the first instruction of the subroutine. This address is placed in the program counter.
		PC + 3 → PC PC_L → S PC_H → S SP - 2 → SP (aaaa) → PC		Extended	JSR $aaaa	BD	9	3	
RTS	ReTurn from Subroutine	S → PC_H S → PC_L SP + 2 → SP	xxxxxx	Implied	RTS	39	5	1	The address of the next instruction in the main program after the last JSR is loaded from the stack into the program counter 1 byte at a time.
BSR	Branch to SubRoutine	PC + 2 → PC PC_L → S PC_H → S SP - 2 → SP PC + rr → PC	xxxxxx	Relative	BSR $rr	8D	8	2	The program counter is incremented by 2 and pushed onto the stack 1 byte at a time. The memory location of the next instruction is then calculate by adding the 2's-complement binary number rr to the program counter. This instruction differs from JSR in the form of addressing it uses.
		Stack Instructions							
LDS	LoaD Stack pointer	M → SP_H (M + 1) → SP_L	xxNZ0x	Immediate Direct Indexed Extended	LDS #$dddd LDS $aa LDS $ff,X LDS $aaaa	8E 9E AE BE	3 4 6 5	3 2 2 3	
STS	STore Stack pointer	SP_H → M SP_L → (M + 1)	xxNZ0x	Direct Indexed Extended	STS $aa STS $ff,X STS $aaaa	9F AF BF	5 7 6	2 2 3	
PSHA	PuSH accumulator A onto the stack	A → S SP - 1 → SP	xxxxxx	Implied	PSHA	36	4	1	Whenever A or B is pushed onto the stack the stack pointer is decremented by 1. When the contents of the stack are placed in A or B the stack pointer is incremented by 1.
PSHB	PuSH accumulator B onto the stack	B → S SP - 1 → SP	xxxxxx	Implied	PSHB	37	4	1	
PULA	PUlL accumulator A from the stack	S → A SP + 1 → SP	xxxxxx	Implied	PULA	32	4	1	
PULB	PUlL accumulator B from the stack	S → B SP + 1 → SP	xxxxxx	Implied	PULB	33	4	1	

EXPANDED TABLE OF 6800 INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Mne-monic	Operation	Boolean/Arith. Operation	Flags HINZVC	Address Mode	Assembler Notation	Op	~	#	Notes
DES	DEcrement Stack pointer	SP - 1 → SP	xxxxxx	Implied	DES	34	4	1	
INS	INcrement Stack pointer	SP + 1 → SP	xxxxxx	Implied	INS	31	4	1	
TXS	Transfer X (index) register to Stack pointer	X - 1 → SP	xxxxxx	Implied	TXS	35	4	1	
TSX	Transfer Stack pointer to the X (index) register	SP + 1 → X	xxxxxx	Implied	TSX	30	4	1	

Interrupt Instructions

Mne-monic	Operation	Boolean/Arith. Operation	Flags HINZVC	Address Mode	Assembler Notation	Op	~	#	Notes
RTI	ReTurn from Interrupt	S → CCR S → B S → A S → X_H S → X_L S → PC_H S → PC_L	HINZVC	Implied	RTI	3B	10	1	
SWI	SoftWare Interrupt	PC + 1 → PC PC_L → S PC_H → S X_L → S X_H → S A → S B → S CCR → S	x1xxxx	Implied	SWI	3F	12	1	After the actions shown in the "Boolean/Arithmetic Operation" column take place, the microprocessor will begin to execute a program whose address is found in memory locations FFFA and FFFB.

Input-Output Instructions

Mne-monic	Operation	Boolean/Arith. Operation	Flags HINZVC	Address Mode	Assembler Notation	Op	~	#	Notes
none									The 6800/6808 has no special input and output instructions but rather memory-maps these operations.

Notes

Addressing Modes	Assembler Notation
Immediate	Mnemonic #$dd
Direct	Mnemonic $aa
Indexed	Mnemonic $ff,X
Extended	Mnemonic $aaaa
Implied	Mnemonic
Relative	Mnemonic $rr

Abbreviations and Explanations

a = address (one hex digit)
d = data (one hex digit)
f = offset (one hex digit) to be added to the X register (ff is positive -- $00-$ff which is decimal 0-255)
r = relative displacement (one hex digit) to be added to the program counter (rr is 2's-complement number and thus can be positive or negative, -128 to +127)
$ = indicates a hexadecimal number
= indicates the data follows immediately after the instruction

L = low byte (lower byte of a two byte number)
H = high byte (upper byte of a two byte number)

Flags

H = instruction affects the half carry-flag
I = instruction affects the interrupt flag
N = instruction affects the negative flag
Z = instruction affects the zero flag
V = instruction affects the overflow flag
C = instruction affects the carry flag
0 = instruction always clears affected flag
1 = instruction always sets affected flag
x = flag not affected by instruction

CCR = condition code register (flags)
S = stack
SP = stack pointer
PC = program counter
() = contents of the memory location in the parenthesis

$M_7...M_0$ = memory bits 0-7 of a particular memory location
$A_7...A_0$ = bits 0-7 of accumulator a
$B_7...B_0$ = bits 0-7 of accumulator b

X = Index register

0 = One zero bit.
00 = One zero byte.

Symbols in the Page Heading

~ = clock cycles
= # of bytes used by instruction (and following address or data if used)

Addressing Modes - Summary

Immediate (Mnemonic #$dd): In this addressing mode, the operand (data or number that something is being done to) is contained in the memory location(s) immediately following the instruction.

Direct (Mnemonic $aa): Direct addressing places the address of the operand in the byte following the instruction.

Indexed (Mnemonic $ff,X): This mode involves a couple of steps. First, the number ff (which is the byte after the instruction) is added to the value in the X register. The number ff is an 8-bit number which can only be positive (0-255 decimal). Then the operand is fetched from this newly formed address.

Extended (Mnemonic $aaaa): Extended addressing is the same as Direct except that a wider range is possible. The first byte is the instruction as in Direct addressing. The second and third bytes then form a 16-bit address where the operand can be found.

Implied (Mnemonic): When the operand is within the microprocessor itself implied addressing is used. In these cases the location of the operand is contained within the instruction itself. CLRA (CLeaR accumulator A) is an example of implied addressing.

Relative (Mnemonic $rr): Relative addressing is used exclusively with the branch and jump instructions. The byte following the instruction is an 8-bit 2's-complement number (+127 to -128) which is added to the contents of the program counter. This then is the address of the next instruction. The location of the next instruction is being indicated relative to the current location in memory (the current contents of the program counter).

SHORT TABLE OF 6800 INSTRUCTIONS LISTED ALPHABETICALLY

Mnemonic	Operation	Assembler Notation	Op
ABA	Add accumulator **B** to accumulator **A**	ABA	1B
ADCA	AdD with Carry accumulator **A** to memory location	ADCA #$dd	89
		ADCA $aa	99
		ADCA $ff,X	A9
		ADCA $aaaa	B9
ADCB	AdD with Carry accumulator **B** to memory location	ADCB #$dd	C9
		ADCB $aa	D9
		ADCB $ff,X	E9
		ADCB $aaaa	F9
ADDA	ADD accumulator **A** to memory location	ADDA #$dd	8B
		ADDA $aa	9B
		ADDA $ff,X	AB
		ADDA $aaaa	BB
ADDB	ADD accumulator **B** to memory location	ADDB #$dd	CB
		ADDB $aa	DB
		ADDB $ff,X	EB
		ADDB $aaaa	FB
ANDA	AND accumulator **A** with memory location	ANDA #$dd	84
		ANDA $aa	94
		ANDA $ff,X	A4
		ANDA $aaaa	B4
ANDB	AND accumulator **B** with memory location	ANDB #$dd	C4
		ANDB $aa	D4
		ANDB $ff,X	E4
		ANDB $aaaa	F4
ASL	Arithmetic Shift Left memory location	ASL $ff,X	68
		ASL $aaaa	78
ASLA	Arithmetic Shift Left accumulator **A**	ASLA	48
ASLB	Arithmetic Shift Left accumulator **B**	ASLB	58
ASR	Arithmetic Shift Right memory location	ASR $ff,X	67
		ASR $aaaa	77
ASRA	Arithmetic Shift Right accumulator **A**	ASRA	47
ASRB	Arithmetic Shift Right accumulator **B**	ASRB	57
BCC	Branch if Carry Clear	BCC $rr	24

Mnemonic	Operation	Assembler Notation	Op
BCS	Branch if Carry Set	BCS $rr	25
BEQ	Branch if result of last operation was EQual to zero	BEQ $rr	27
BGE	Branch if Greater than or Equal to zero	BGE $rr	2C
BGT	Branch if Greater Than zero	BGT $rr	2E
BHI	Branch if HIgher	BHI $rr	22
BITA	BIT test accumulator **A**	BITA #$dd	85
		BITA $aa	95
		BITA $ff,X	A5
		BITA $aaaa	B5
BITB	BIT test accumulator **B**	BITB #$dd	C5
		BITB $aa	D5
		BITB $ff,X	E5
		BITB $aaaa	F5
BLE	Branch if Less then or Equal to zero	BLE $rr	2F
BLS	Branch if Lower or the Same	BLS $rr	23
BLT	Branch if Less Than zero	BLT $rr	2D
BMI	Branch is MInus	BMI $rr	2B
BNE	Branch if Not Equal to zero	BNE $rr	26
BPL	Branch if PLus	BPL $rr	2A
BRA	BRanch Always to memory location	BRA $rr	20
BSR	Branch to SubRoutine	BSR $rr	8D
BVC	Branch if oVerflow Clear	BVC $rr	28
BVS	Branch if oVerflow Set	BVS $rr	29

Mnemonic	Operation	Assembler Notation	Op
CBA	Compare accumulator **B** to accumulator **A**	CBA	11
CLC	CLear Carry flag	CLC	0C
CLI	CLear Interrupt flag	CLI	0E
CLR	CLeaR memory location	CLR $ff,X	6F
		CLR $aaaa	7F
CLRA	CLeaR accumulator **A**	CLRA	4F
CLRB	CLeaR accumulator **B**	CLRB	5F
CLV	CLear oVerflow flag	CLV	0A
CMPA	CoMPare memory location to accumulator **A**	CMPA #$dd	81
		CMPA $aa	91
		CMPA $ff,X	A1
		CMPA $aaaa	B1
CMPB	CoMPare memory location to accumulator **B**	CMPB #$dd	C1
		CMPB $aa	D1
		CMPB $ff,X	E1
		CMPB $aaaa	F1
COM	COMplement memory location (1's complement)	COM $ff,X	63
		COM $aaaa	73
COMA	COMplement accumulator **A** (1's complement)	COMA	43
COMB	COMplement accumulator **B** (1's complement)	COMB	53
CPX	ComPare memory location to **X** (index) register	CPX #$dd	8C
		CPX $aa	9C
		CPX $ff,X	AC
		CPX $aaaa	BC
DAA	Decimal Adjust accumulator **A**	DAA	19
DEC	DECrement memory location	DEC $ff,X	6A
		DEC $aaaa	7A
DECA	DECrement accumulator **A**	DECA	4A
DECB	DECrement accumulator **B**	DECB	5A
DES	DEcrement Stack pointer	DES	34

Mnemonic	Operation	Assembler Notation	Op
DEX	DEcrement **X** (index) register	DEX	09
EORA	Exclusively **OR** accumulator **A** with memory location	EORA #$dd	88
		EORA $aa	98
		EORA $ff,X	A8
		EORA $aaaa	B8
EORB	Exclusively **OR** accumulator **A** with memory location	EORB #$dd	C8
		EORB $aa	D8
		EORB $ff,X	E8
		EORB $aaaa	F8
INC	INCrement memory location	INC $ff,X	6C
		INC $aaaa	7C
INCA	INCrement accumulator **A**	INCA	4C
INCB	INCrement accumulator **B**	INCB	5C
INS	INcrement Stack pointer	INS	31
INX	INcrement **X** (index) register	INX	08
JMP	JuMP to memory location	JMP $ff,X	6E
		JMP $aaaa	7E
JSR	Jump SubRoutine	JSR $ff,X	AD
		JSR $aaaa	BD
LDAA	LoaD Accumulator A	LDAA #$dd	86
		LDAA $aa	96
		LDAA $ff,X	A6
		LDAA $aaaa	B6
LDAB	LoaD Accumulator **B**	LDAB #$dd	C6
		LDAB $aa	D6
		LDAB $ff,X	E6
		LDAB $aaaa	F6
LDS	LoaD Stack pointer	LDS #$dddd	8E
		LDS $aa	9E
		LDS $ff,X	AE
		LDS $aaaa	BE
LDX	LoaD **X** register	LDX #$dd	CE
		LDX $aa	DE
		LDX $ff,X	EE
		LDX $aaaa	FE
LSR	Logical Shift Right memory location	LSR $ff,X	64
		LSR $aaaa	74

SHORT TABLE OF 6800 INSTRUCTIONS LISTED ALPHABETICALLY *(Continued)*

Mne-monic	Operation	Assembler Notation	Op
LSRA	Logical Shift Right accumulator A	LSRA	44
LSRB	Logical Shift Right accumulator B	LSRB	54
NEG	NEGate memory location (2's complement)	NEG $ff,X	60
		NEG $aaaa	70
NEGA	NEGate accumulator A (2's complement)	NEGA	40
NEGB	NEGate accumulator B (2's complement)	NEGB	50
NOP	No OPeration	NOP	01
ORAA	OR Accumulator A with memory location	ORAA #$dd	8A
		ORAA $aa	9A
		ORAA $ff,X	AA
		ORAA $aaaa	BA
ORAB	OR Accumulator B with memory location	ORAB #$dd	CA
		ORAB $aa	DA
		ORAB $ff,X	EA
		ORAB $aaaa	FA
PSHA	PuSH accumulator A onto the stack	PSHA	36
PSHB	PuSH accumulator B onto the stack	PSHB	37
PULA	PUlL accumulator A from the stack	PULA	32
PULB	PUlL accumulator B from the stack	PULB	33
ROL	ROtate memory location Left	ROL $ff,X	69
		ROL $aaaa	79
ROLA	ROtate to the Left accumulator A	ROLA	49
ROLB	ROtate to the Left accumulator B	ROLB	59
ROR	ROtate memory location Right	ROR $ff,X	66
		ROR $aaaa	76
RORA	ROtate to the Right accumulator A	RORA	46
RORB	ROtate to the Right accumulator B	RORB	56
RTI	ReTurn from Interrupt	RTI	3B
RTS	ReTurn from Subroutine	RTS	39
SBA	Subtract accumulator B from accumulator A	SBA	10
SBCA	SuBtract with Carry memory location from accumulator A	SBCA #$dd	82
		SBCA $aa	92
		SBCA $ff,X	A2
		SBCA $aaaa	B2
SBCB	SuBtract with Carry memory location from accumulator B	SBCB #$dd	C2
		SBCB $aa	D2
		SBCB $ff,X	E2
		SBCB $aaaa	F2
SEC	SEt Carry flag	SEC	0D
SEI	SEt Interrupt flag	SEI	0F
SEV	SEt oVerflow flag	SEV	0B
STAA	STore Accumulator A	STAA $aa	97
		STAA $ff,X	A7
		STAA $aaaa	B7
STAB	STore Accumulator B	STAB $aa	D7
		STAB $ff,X	E7
		STAB $aaaa	F7
STS	STore Stack pointer	STS $aa	9F
		STS $ff,X	AF
		STS $aaaa	BF
STX	STore X register	STX $aa	DF
		STX $ff,X	EF
		STX $aaaa	FF
SUBA	SUBtract memory location from accumulator A	SUBA #$dd	80
		SUBA $aa	90
		SUBA $ff,X	A0
		SUBA $aaaa	B0
SUBB	SUBtract memory location from accumulator B	SUBB #$dd	C0
		SUBB $aa	D0
		SUBB $ff,X	E0
		SUBB $aaaa	F0
SWI	SoftWare Interrupt	SWI	3F

Mnemonic	Operation	Assembler Notation	Op
TAB	Transfer A to B	TAB	16
TAP	Transfer Accumulator A to Processor condition code register	TAP	06
TBA	Transfer B to A	TBA	17
TPA	Transfer Processor condition code register to accumulator A	TPA	07
TST	TEsT memory location for zero or minus	TST $ff,X TST $aaaa	6D 7D

Mnemonic	Operation	Assembler Notation	Op
TSTA	TEsT accumulator A for zero or minus	TSTA	4D
TSTB	TEsT accumulator B for zero or minus	TSTB	5D
TSX	Transfer Stack pointer to the X (index) register	TSX	30
TXS	Transfer X (index) register to Stack pointer	TXS	35
WAI	WAit for Interrupt	WAI	3E

SHORT TABLE OF 6800 INSTRUCTIONS LISTED BY CATEGORY

CPU Control Instructions

Assembler Notation	Op	Boolean/Arith Operation	Flags HINZVC
NOP	01	nothing	xxxxxx
WAI	3E	PC + 1 → PC PC_L → S PC_H → S X_L → S X_H → S A → S B → S CCR → S	xIxxxx

Data Transfer Instructions

Assembler Notation	Op	Boolean/Arith Operation	Flags HINZVC
LDAA #$dd LDAA $aa LDAA $ff,X LDAA $aaaa	86 96 A6 B6	M → A	xxNZ0x
LDAB #$dd LDAB $aa LDAB $ff,X LDAB $aaaa	C6 D6 E6 F6	M → B	xxNZ0x
STAA $aa STAA $ff,X STAA $aaaa	97 A7 B7	A → M	xxNZ0x
STAB $aa STAB $ff,X STAB $aaaa	D7 E7 F7	B → M	xxNZ0x
TAB	16	A → B	xxNZ0x
TBA	17	B → A	xxNZ0x
LDX #$dddd LDX $aa LDX $ff,X LDX $aaaa	CE DE EE FE	M → X_H (M + 1) → X_L	xxNZ0x
STX $aa STX $ff,X STX $aaaa	DF EF FF	X_H → M X_L → (M + 1)	xxNZ0x
CLR $ff,X CLR $aaaa	6F 7F	00 → M	xx0100
CLRA	4F	00 → A	xx0100
CLRB	5F	00 → B	xx0100

Flag Instructions

Assembler Notation	Op	Boolean/Arith Operation	Flags HINZVC
CLC	0C	0 → C	xxxxx0
LI	0E	0 → I	x0xxxx
CLV	0A	0 → V	xxxxVx
SEC	0D	1 → C	xxxxx1
SEI	0F	1 → I	x1xxxx

SHORT TABLE OF 6800 INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Assembler Notation	Op	Boolean/Arith Operation	Flags HINZVC
SEV	0B	1 → V	xxxx1x
TAP	06	A → CCR	HINZVC
TPA	07	CCR → A	xxxxxx

Arithmetic Instructions

Assembler Notation	Op	Boolean/Arith Operation	Flags HINZVC
ADDA #$dd	8B	A + M → A	HxNZVC
ADDA $aa	9B		
ADDA $ff,X	AB		
ADDA $aaaa	BB		
ADDB #$dd	CB	B + M → B	HxNZVC
ADDB $aa	DB		
ADDB $ff,X	EB		
ADDB $aaaa	FB		
ABA	1B	A + B → A	HxNZVC
ADCA #$dd	89	A + M + C → A	HxNZVC
ADCA $aa	99		
ADCA $ff,X	A9		
ADCA $aaaa	B9		
ADCB #$dd	C9	B + M + C → B	HxNZVC
ADCB $aa	D9		
ADCB $ff,X	E9		
ADCB $aaaa	F9		
SUBA #$dd	80	A - M → A	xxNZVC
SUBA $aa	90		
SUBA $ff,X	A0		
SUBA $aaaa	B0		
SUBB #$dd	C0	B - M → B	xxNZVC
SUBB $aa	D0		
SUBB $ff,X	E0		
SUBB $aaaa	F0		
SBA	10	A - B → A	xxNZVC
SBCA #$dd	82	A - M - C → A	xxNZVC
SBCA $aa	92		
SBCA $ff,X	A2		
SBCA $aaaa	B2		
SBCB #$dd	C2	B - M - C → B	xxNZVC
SBCB $aa	D2		
SBCB $ff,X	E2		
SBCB $aaaa	F2		

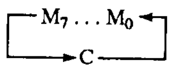
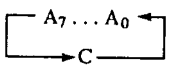
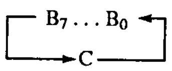
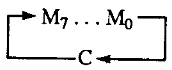
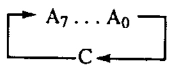
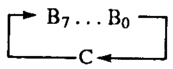
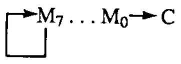
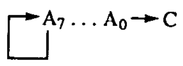
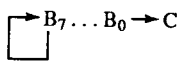
Assembler Notation	Op	Boolean/Arith Operation	Flags HINZVC
DAA	19	(converts binary add. of BCD characters into BCD format)	xxNZVC

Logical Instructions

Assembler Notation	Op	Boolean/Arith Operation	Flags HINZVC
ANDA #$dd	84	A AND M → A	xxNZ0x
ANDA $aa	94		
ANDA $ff,X	A4		
ANDA $aaaa	B4		
ANDB #$dd	C4	B AND M → B	xxNZ0x
ANDB $aa	D4		
ANDB $ff,X	E4		
ANDB $aaaa	F4		
ORAA #$dd	8A	A OR M → A	xxNZ0x
ORAA $aa	9A		
ORAA $ff,X	AA		
ORAA $aaaa	BA		
ORAB #$dd	CA	B OR M → B	xxNZ0x
ORAB $aa	DA		
ORAB $ff,X	EA		
ORAB $aaaa	FA		
EORA #$dd	88	A EOR M → A	xxNZ0x
EORA $aa	98		
EORA $ff,X	A8		
EORA $aaaa	B8		
EORB #$dd	C8	B EOR M → B	xxNZ0x
EORB $aa	D8		
EORB $ff,X	E8		
EORB $aaaa	F8		
BITA #$dd	85	A AND M	xxNZ0x
BITA $aa	95		
BITA $ff,X	A5		
BITA $aaaa	B5		
BITB #$dd	C5	B AND M	xxNZ0x
BITB $aa	D5		
BITB $ff,X	E5		
BITB $aaaa	F5		
COM $ff,X	63	$\overline{M} \rightarrow M$	xxNZ01
COM $aaaa	73		
COMA	43	$\overline{A} \rightarrow A$	xxNZ01
COMB	53	$\overline{B} \rightarrow B$	xxNZ01

Assembler Notation	Op	Boolean/Arith Operation	Flags HINZVC
NEG $ff,X NEG $aaaa	60 70	00 - M → M	xxNZVC
NEGA	40	00 - A → A	xxNZVC
NEGB	50	00 - B → B	xxNZVC

Rotate and Shift Instructions

Assembler Notation	Op	Boolean/Arith Operation	Flags HINZVC
ROL $ff,X ROL $aaaa	69 79		xxNZVC
ROLA	49		xxNZVC
ROLB	59		xxNZVC
ROR $ff,X ROR $aaaa	66 76		xxNZVC
RORA	46		xxNZVC
RORB	56		xxNZVC
ASL $ff,X ASL $aaaa	68 78	$C \leftarrow M_7 \ldots M_0 \leftarrow 0$	xxNZVC
ASLA	48	$C \leftarrow A_7 \ldots A_0 \leftarrow 0$	xxNZVC
ASLB	58	$C \leftarrow B_7 \ldots B_0 \leftarrow 0$	xxNZVC
ASR $ff,X ASR $aaaa	67 77		xxNZVC
ASRA	47		xxNZVC
ASRB	57		xxNZVC
LSR $ff,X LSR $aaaa	64 74	$0 \rightarrow M_7 \ldots M_0 \rightarrow C$	xx0ZVC
LSRA	44	$0 \rightarrow A_7 \ldots A_0 \rightarrow C$	xx0ZVC
LSRB	54	$0 \rightarrow B_7 \ldots B_0 \rightarrow C$	xx0ZVC

Increment and Decrement Instructions

Assembler Notation	Op	Boolean/Arith Operation	Flags HINZVC
INC $ff,X INC $aaaa	6C 7C	M + 1 → M	xxNZVx
INCA	4C	A + 1 → A	xxNZVx
INCB	5C	B + 1 → B	xxNZVx
DEC $ff,X DEC $aaaa	6A 7A	M - 1 → M	xxNZVx
DECA	4A	A - 1 → A	xxNZVx
DECB	5A	B - 1 → B	xxNZVx
INX	08	X + 1 → X	xxxZxx
DEX	09	X - 1 → X	xxxZxx

Unconditional Jump Instructions

Assembler Notation	Op	Boolean/Arith Operation	Flags HINZVC
JMP $ff,X JMP $aaaa	6E 7E	X + ff → PC (indexed) aaaa → PC (extended)	xxxxxx
BRA $rr	20	PC + 2 + rr → PC	xxxxxx

Test (Compare) Instructions

Assembler Notation	Op	Boolean/Arith Operation	Flags HINZVC
CMPA #$dd CMPA $aa CMPA $ff,X CMPA $aaaa	81 91 A1 B1	A - M	xxNZVC
CMPB #$dd CMPB $aa CMPB $ff,X CMPB $aaaa	C1 D1 E1 F1	B - M	xxNZVC
CBA	11	A - B	xxNZVC
CPX #$dddd CPX $aa CPX $ff,X CPX $aaaa	8C 9C AC BC	X_H - M X_L - (M+1)	xxNZVx
TST $ff,X TST $aaaa	6D 7D	M - 00	xxNZ00
TSTA	4D	A - 00	xxNZ00
TSTB	5D	B - 00	xxNZ00

SHORT TABLE OF 6800 INSTRUCTIONS LISTED BY CATEGORY (*Continued*)

Conditional Jump (Branch) Instructions

Assembler Notation	Op	Boolean/Arith Operation	Flags HINZVC
BCC \$rr	24	PC + 2 + rr → PC If C=0	xxxxxx
BCS \$rr	25	PC + 2 + rr → PC If C=1	xxxxxx
BEQ \$rr	27	PC + 2 + rr → PC If Z=1	xxxxxx
BGE \$rr	2C	PC + 2 + rr → PC If N EOR V = 0	xxxxxx
BGT \$rr	2E	PC + 2 + rr → PC If Z AND (N EOR V) = 0	xxxxxx
BHI \$rr	22	PC + 2 + rr → PC If C AND Z = 0	xxxxxx
BLE \$rr	2F	PC + 2 + rr → PC If Z AND (N EOR V) = 1	xxxxxx
BLS \$rr	23	PC + 2 + rr → PC If C OR Z = 1	xxxxxx
BLT \$rr	2D	PC + 2 + rr → PC If N EOR V = 1	xxxxxx
BMI \$rr	2B	PC + 2 + rr → PC If N=1	xxxxxx
BNE \$rr	26	PC + 2 + rr → PC If Z=1	xxxxxx
BVC \$rr	28	PC + 2 + rr → PC If V=0	xxxxxx
BVS \$rr	29	PC + 2 + rr → PC If V=1	xxxxxx
BPL \$rr	2A	PC + 2 + rr → PC If N=0	xxxxxx

Subroutine Instructions

Assembler Notation	Op	Boolean/Arith Operation	Flags HINZVC
JSR \$ff,X	AD	PC + 2 → PC PC_L → S PC_H → S SP - 2 → SP (ff+X) → PC	xxxxxx
JSR \$aaaa	BD	PC + 3 → PC PC_L → S PC_H → S SP - 2 → SP (aaaa) → PC	
RTS	39	S → PC_H S → PC_L SP + 2 → SP	xxxxxx
BSR \$rr	8D	PC + 2 → PC PC_L → S PC_H → S SP - 2 → SP PC + rr → PC	xxxxxx

Stack Instructions

Assembler Notation	Op	Boolean/Arith Operation	Flags HINZVC
LDS #\$dddd	8E	M → SP_H	xxNZ0x
LDS \$aa	9E	(M + 1) → SP_L	
LDS \$ff,X	AE		
LDS \$aaaa	BE		
STS \$aa	9F	SP_H → M	xxNZ0x
STS \$ff,X	AF	SP_L → (M + 1)	
STS \$aaaa	BF		
PSHA	36	A → S SP - 1 → SP	xxxxxx
PSHB	37	B → S SP - 1 → SP	xxxxxx
PULA	32	S → A SP + 1 → SP	xxxxxx
PULB	33	S → B SP + 1 → SP	xxxxxx
DES	34	SP - 1 → SP	xxxxxx
INS	31	SP + 1 → SP	xxxxxx

Assembler Notation	Op	Boolean/Arith Operation	Flags HINZVC
TXS	35	$X - 1 \rightarrow SP$	xxxxxx
TSX	30	$SP + 1 \rightarrow X$	xxxxxx

Interrupt Instructions

Assembler Notation	Op	Boolean/Arith Operation	Flags HINZVC
RTI	3B	$S \rightarrow CCR$ $S \rightarrow B$ $S \rightarrow A$ $S \rightarrow X_H$ $S \rightarrow X_L$ $S \rightarrow PC_H$ $S \rightarrow PC_L$	HINZVC

Assembler Notation	Op	Boolean/Arith Operation	Flags HINZVC
SWI	3F	$PC + 1 \rightarrow PC$ $PC_L \rightarrow S$ $PC_H \rightarrow S$ $X_L \rightarrow S$ $X_H \rightarrow S$ $A \rightarrow S$ $B \rightarrow S$ $CCR \rightarrow S$	x1xxxx

Input-Output Instructions

none

CONDENSED TABLE OF 6800 INSTRUCTIONS LISTED BY CATEGORY

Assembler	Op
CPU Control Instructions	
NOP	01
WAI	3E
Data Transfer Instructions	
LDAA #$dd	86
LDAA $aa	96
LDAA $ff,X	A6
LDAA $aaaa	B6
LDAB #$dd	C6
LDAB $aa	D6
LDAB $ff,X	E6
LDAB $aaaa	F6
STAA $aa	97
STAA $ff,X	A7
STAA $aaaa	B7
STAB $aa	D7
STAB $ff,X	E7
STAB $aaaa	F7
TAB	16
TBA	17
LDX #$dddd	CE
LDX $aa	DE
LDX $ff,X	EE
LDX $aaaa	FE

Assembler	Op
STX $aa	DF
STX $ff,X	EF
STX $aaaa	FF
CLR $ff,X	6F
CLR $aaaa	7F
CLRA	4F
CLRB	5F
Flag Instructions	
CLC	0C
LI	0E
CLV	0A
SEC	0D
SEI	0F
SEV	0B
TAP	06
TPA	07
Arithmetic Instructions	
ADDA #$dd	8B
ADDA $aa	9B
ADDA $ff,X	AB
ADDA $aaaa	BB
ADDB #$dd	CB
ADDB $aa	DB
ADDB $ff,X	EB
ADDB $aaaa	FB
ABA	1B

Assembler	Op
ADCA #$dd	89
ADCA $aa	99
ADCA $ff,X	A9
ADCA $aaaa	B9
ADCB #$dd	C9
ADCB $aa	D9
ADCB $ff,X	E9
ADCB $aaaa	F9
SUBA #$dd	80
SUBA $aa	90
SUBA $ff,X	A0
SUBA $aaaa	B0
SUBB #$dd	C0
SUBB $aa	D0
SUBB $ff,X	E0
SUBB $aaaa	F0
SBA	10
SBCA #$dd	82
SBCA $aa	92
SBCA $ff,X	A2
SBCA $aaaa	B2
SBCB #$dd	C2
SBCB $aa	D2
SBCB $ff,X	E2
SBCB $aaaa	F2
DAA	19

Assembler	Op
Logical Instructions	
ANDA #$dd	84
ANDA $aa	94
ANDA $ff,X	A4
ANDA $aaaa	B4
ANDB #$dd	C4
ANDB $aa	D4
ANDB $ff,X	E4
ANDB $aaaa	F4
ORAA #$dd	8A
ORAA $aa	9A
ORAA $ff,X	AA
ORAA $aaaa	BA
ORAB #$dd	CA
ORAB $aa	DA
ORAB $ff,X	EA
ORAB $aaaa	FA
EORA #$dd	88
EORA $aa	98
EORA $ff,X	A8
EORA $aaaa	B8
EORB #$dd	C8
EORB $aa	D8
EORB $ff,X	E8
EORB $aaaa	F8

CONDENSED TABLE OF 6800 INSTRUCTIONS LISTED BY CATEGORY (*Continued*)

Assembler	Op
BITA #$dd	85
BITA $aa	95
BITA $ff,X	A5
BITA $aaaa	B5
BITB #$dd	C5
BITB $aa	D5
BITB $ff,X	E5
BITB $aaaa	F5
COM $ff,X	63
COM $aaaa	73
COMA	43
COMB	53
NEG $ff,X	60
NEG $aaaa	70
NEGA	40
NEGB	50
Rotate and Shift Instructions	
ROL $ff,X	69
ROL $aaaa	79
ROLA	49
ROLB	59
ROR $ff,X	66
ROR $aaaa	76
RORA	46
RORB	56
ASL $ff,X	68
ASL $aaaa	78
ASLA	48
ASLB	58
ASR $ff,X	67
ASR $aaaa	77
ASRA	47
ASRB	57
LSR $ff,X	64
LSR $aaaa	74
LSRA	44
LSRB	54
Increment and Decrement Instructions	
INC $ff,X	6C
INC $aaaa	7C
INCA	4C
INCB	5C
DEC $ff,X	6A
DEC $aaaa	7A
DECA	4A
DECB	5A
INX	08
DEX	09
Unconditional Jump Instructions	
JMP $ff,X	6E
JMP $aaaa	7E
BRA $rr	20
Test (Compare) Instructions	
CMPA #$dd	81
CMPA $aa	91
CMPA $ff,X	A1
CMPA $aaaa	B1
CMPB #$dd	C1
CMPB $aa	D1
CMPB $ff,X	E1
CMPB $aaaa	F1
CBA	11
CPX #$dddd	8C
CPX $aa	9C
CPX $ff,X	AC
CPX $aaaa	BC
TST $ff,X	6D
TST $aaaa	7D
TSTA	4D
TSTB	5D
Conditional Jump (Branch) Instructions	
BCC $rr	24
BCS $rr	25
BEQ $rr	27
BGE $rr	2C
BGT $rr	2E
BHI $rr	22
BLE $rr	2F
BLS $rr	23
BLT $rr	2D
BMI $rr	2B
BNE $rr	26
BVC $rr	28
BVS $rr	29
BPL $rr	2A
Subroutine Instructions	
JSR $ff,X	AD
JSR $aaaa	BD
RTS	39
BSR $rr	8D
Stack Instructions	
LDS #$dddd	8E
LDS $aa	9E
LDS $ff,X	AE
LDS $aaaa	BE
STS $aa	9F
STS $ff,X	AF
STS $aaaa	BF
PSHA	36
PSHB	37
PULA	32
PULB	33
DES	34
INS	31
TXS	35
TSX	30
Interrupt Instructions	
RTI	3B
SWI	3F
Input-Output Instructions	
none	

CONDENSED TABLE OF 6800 INSTRUCTIONS LISTED ALPHABETICALLY

Assembler	Op	Assembler	Op	Assembler	Op	Assembler	Op
ABA	1B	BMI $rr	2B	INS	31	RORB	56
ADCA $aa	99	BNE $rr	26	INX	08	RTI	3B
ADCA $aaaa	B9	BPL $rr	2A	JMP $aaaa	7E	RTS	39
ADCA $ff,X	A9	BRA $rr	20	JMP $ff,X	6E	SBA	10
ADCA #$dd	89	BSR $rr	8D	JSR $aaaa	BD	SBCA $aa	92
ADCB $aa	D9	BVC $rr	28	JSR $ff,X	AD	SBCA $aaaa	B2
ADCB $aaaa	F9	BVS $rr	29	LDAA $aa	96	SBCA $ff,X	A2
ADCB $ff,X	E9	CBA	11	LDAA $aaaa	B6	SBCA #$dd	82
ADCB #$dd	C9	CLC	0C	LDAA $ff,X	A6	SBCB $aa	D2
ADDA $aa	9B	CLI	0E	LDAA #$dd	86	SBCB $aaaa	F2
ADDA $aaaa	BB	CLR $aaaa	7F	LDAB $aa	D6	SBCB $ff,X	E2
ADDA $ff,X	AB	CLR $ff,X	6F	LDAB $aaaa	F6	SBCB #$dd	C2
ADDA #$dd	8B	CLRA	4F	LDAB $ff,X	E6	SEC	0D
ADDB $aa	DB	CLRB	5F	LDAB #$dd	C6	SEI	0F
ADDB $aaaa	FB	CLV	0A	LDS $aa	9E	SEV	0B
ADDB $ff,X	EB	CMPA $aa	91	LDS $aaaa	BE	STAA $aa	97
ADDB #$dd	CB	CMPA $aaaa	B1	LDS $ff,X	AE	STAA $aaaa	B7
ANDA $aa	94	CMPA $ff,X	A1	LDS #$dddd	8E	STAA $ff,X	A7
ANDA $aaaa	B4	CMPA #$dd	81	LDX $aa	DE	STAB $aa	D7
ANDA $ff,X	A4	CMPB $aa	D1	LDX $aaaa	FE	STAB $aaaa	F7
ANDA #$dd	84	CMPB $aaaa	F1	LDX $ff,X	EE	STAB $ff,X	E7
ANDB $aa	D4	CMPB $ff,X	E1	LDX #$dd	CE	STS $aa	9F
ANDB $aaaa	F4	CMPB #$dd	C1	LSR $aaaa	74	STS $aaaa	BF
ANDB $ff,X	E4	COM $aaaa	73	LSR $ff,X	64	STS $ff,X	AF
ANDB #$dd	C4	COM $ff,X	63	LSRA	44	STX $aa	DF
ASL $aaaa	78	COMA	43	LSRB	54	STX $aaaa	FF
ASL $ff,X	68	COMB	53	NEG $aaaa	70	STX $ff,X	EF
ASLA	48	CPX $aa	9C	NEG $ff,X	60	SUBA $aa	90
ASLB	58	CPX $aaaa	BC	NEGA	40	SUBA $aaaa	B0
ASR $aaaa	77	CPX $ff,X	AC	NEGB	50	SUBA $ff,X	A0
ASR $ff,X	67	CPX #$dd	8C	NOP	01	SUBA #$dd	80
ASRA	47	DAA	19	ORAA $aa	9A	SUBB $aa	D0
ASRB	57	DEC $aaaa	7A	ORAA $aaaa	BA	SUBB $aaaa	F0
BCC $rr	24	DEC $ff,X	6A	ORAA $ff,X	AA	SUBB $ff,X	E0
BCS $rr	25	DECA	4A	ORAA #$dd	8A	SUBB #$dd	C0
BEQ $rr	27	DECB	5A	ORAB $aa	DA	SWI	3F
BGE $rr	2C	DES	34	ORAB $aaaa	FA	TAB	16
BGT $rr	2E	DEX	09	ORAB $ff,X	EA	TAP	06
BHI $rr	22	EORA $aa	98	ORAB #$dd	CA	TBA	17
BITA $aa	95	EORA $aaaa	B8	PSHA	36	TPA	07
BITA $aaaa	B5	EORA $ff,X	A8	PSHB	37	TST $aaaa	7D
BITA $ff,X	A5	EORA #$dd	88	PULA	32	TST $ff,X	6D
BITA #$dd	85	EORB $aa	D8	PULB	33	TSTA	4D
BITB $aa	D5	EORB $aaaa	F8	ROL $aaaa	79	TSTB	5D
BITB $aaaa	F5	EORB $ff,X	E8	ROL $ff,X	69	TSX	30
BITB $ff,X	E5	EORB #$dd	C8	ROLA	49	TXS	35
BITB #$dd	C5	INC $aaaa	7C	ROLB	59	WAI	3E
BLE $rr	2F	INC $ff,X	6C	ROR $aaaa	76		
BLS $rr	23	INCA	4C	ROR $ff,X	66		
BLT $rr	2D	INCB	5C	RORA	46		

CONDENSED TABLE OF 6800 INSTRUCTIONS LISTED BY OP CODE

Op	Assembler	Op	Assembler	Op	Assembler	Op	Assembler
01	NOP	49	ROLA	8C	CPX #$dd	C4	ANDB #$dd
06	TAP	4A	DECA	8D	BSR $rr	C5	BITB #$dd
07	TPA	4C	INCA	8E	LDS #$dddd	C6	LDAB #$dd
08	INX	4D	TSTA	90	SUBA $aa	C8	EORB #$dd
09	DEX	4F	CLRA	91	CMPA $aa	C9	ADCB #$dd
0A	CLV	50	NEGB	92	SBCA $aa	CA	ORAB #$dd
0B	SEV	53	COMB	94	ANDA $aa	CB	ADDB #$dd
0C	CLC	54	LSRB	95	BITA $aa	CE	LDX #$dd
0D	SEC	56	RORB	96	LDAA $aa	D0	SUBB $aa
0E	CLI	57	ASRB	97	STAA $aa	D1	CMPB $aa
0F	SEI	58	ASLB	98	EORA $aa	D2	SBCB $aa
10	SBA	59	ROLB	99	ADCA $aa	D4	ANDB $aa
11	CBA	5A	DECB	9A	ORAA $aa	D5	BITB $aa
16	TAB	5C	INCB	9B	ADDA $aa	D6	LDAB $aa
17	TBA	5D	TSTB	9C	CPX $aa	D7	STAB $aa
19	DAA	5F	CLRB	9E	LDS $aa	D8	EORB $aa
1B	ABA	60	NEG $ff,X	9F	STS $aa	D9	ADCB $aa
20	BRA $rr	63	COM $ff,X	A0	SUBA $ff,X	DA	ORAB $aa
22	BHI $rr	64	LSR $ff,X	A1	CMPA $ff,X	DB	ADDB $aa
23	BLS $rr	66	ROR $ff,X	A2	SBCA $ff,X	DE	LDX $aa
24	BCC $rr	67	ASR $ff,X	A4	ANDA $ff,X	DF	STX $aa
25	BCS $rr	68	ASL $ff,X	A5	BITA $ff,X	E0	SUBB $ff,X
26	BNE $rr	69	ROL $ff,X	A6	LDAA $ff,X	E1	CMPB $ff,X
27	BEQ $rr	6A	DEC $ff,X	A7	STAA $ff,X	E2	SBCB $ff,X
28	BVC $rr	6C	INC $ff,X	A8	EORA $ff,X	E4	ANDB $ff,X
29	BVS $rr	6D	TST $ff,X	A9	ADCA $ff,X	E5	BITB $ff,X
2A	BPL $rr	6E	JMP $ff,X	AA	ORAA $ff,X	E6	LDAB $ff,X
2B	BMI $rr	6F	CLR $ff,X	AB	ADDA $ff,X	E7	STAB $ff,X
2C	BGE $rr	70	NEG $aaaa	AC	CPX $ff,X	E8	EORB $ff,X
2D	BLT $rr	73	COM $aaaa	AD	JSR $ff,X	E9	ADCB $ff,X
2E	BGT $rr	74	LSR $aaaa	AE	LDS $ff,X	EA	ORAB $ff,X
2F	BLE $rr	76	ROR $aaaa	AF	STS $ff,X	EB	ADDB $ff,X
30	TSX	77	ASR $aaaa	B0	SUBA $aaaa	EE	LDX $ff,X
31	INS	78	ASL $aaaa	B1	CMPA $aaaa	EF	STX $ff,X
32	PULA	79	ROL $aaaa	B2	SBCA $aaaa	F0	SUBB $aaaa
33	PULB	7A	DEC $aaaa	B4	ANDA $aaaa	F1	CMPB $aaaa
34	DES	7C	INC $aaaa	B5	BITA $aaaa	F2	SBCB $aaaa
35	TXS	7D	TST $aaaa	B6	LDAA $aaaa	F4	ANDB $aaaa
36	PSHA	7E	JMP $aaaa	B7	STAA $aaaa	F5	BITB $aaaa
37	PSHB	7F	CLR $aaaa	B8	EORA $aaaa	F6	LDAB $aaaa
39	RTS	80	SUBA #$dd	B9	ADCA $aaaa	F7	STAB $aaaa
3B	RTI	81	CMPA #$dd	BA	ORAA $aaaa	F8	EORB $aaaa
3E	WAI	82	SBCA #$dd	BB	ADDA $aaaa	F9	ADCB $aaaa
3F	SWI	84	ANDA #$dd	BC	CPX $aaaa	FA	ORAB $aaaa
40	NEGA	85	BITA #$dd	BD	JSR $aaaa	FB	ADDB $aaaa
43	COMA	86	LDAA #$dd	BE	LDS $aaaa	FE	LDX $aaaa
44	LSRA	88	EORA #$dd	BF	STS $aaaa	FF	STX $aaaa
46	RORA	89	ADCA #$dd	C0	SUBB #$dd		
47	ASRA	8A	ORAA #$dd	C1	CMPB #$dd		
48	ASLA	8B	ADDA #$dd	C2	SBCB #$dd		

EXPANDED TABLE OF 8086/8088 INSTRUCTIONS LISTED BY CATEGORY

CPU Control Instructions

ESC

ESCape

The ESC instruction allows the 8086/8088 to pass instructions to the 8087 math coprocessor. The instructions for the coprocessor appear as a 6-bit code embedded in the escape instruction. The 8086/8088 performs a NOP while the 8087 executes the instruction. [Flags affected - none]

HLT

HaLT

The HLT instruction causes the 8086/8088 to stop fetching and executing instructions and enter a halt state. To exit from the halt state the microprocessor must receive a hardware reset or interrupt signal. [Flags affected - none]

LOCK

LOCK

LOCK is a prefix which can be used in front of 8086/8088 instructions. It prevents any other processors from gaining access to the systems buses during the following instruction. [Flags affected - none]

NOP

No OPeration

The NOP instruction simply uses up three clock cycles during which nothing is done and no flags are affected. It is useful 1) in programs requiring time delays, and 2) as a means to hold space open in programs so instructions can be added at a later date. [Flags affected - none]

WAIT

WAIT

The WAIT instruction causes the 8086/8088 to enter a wait state or idle condition during which no further processing occurs (except valid interrupts) until a signal is received on the TEST pin. [Flags affected - none]

Data Transfer Instructions

LAHF

Load AH from Flag

The LAHF instruction copies the low-order byte of the flag (status) register to AH. The flags themselves are not affected. The low order byte of the 8086/8088 status register is the same as that of the 8085. This instruction is used primarily to translate 8085 software into 8086/8088 software. [Flags affected - none]

LDS

Load Data Segment

The LDS instruction performs two distinct operations. First it loads two consecutive bytes of memory into one of the 16-bit general, index, or pointer registers. Then it loads the next two consecutive bytes of memory into the 16-bit DS register.

For example, if DI = 1000 then:

```
LDS BX,[DI]
```

copies the contents of memory locations 1000 and 1001 of the data segment into register BX and the contents of memory locations 1002 and 1003 of the data segment into register DS.
[Flags affected - none]

LEA

Load Effective Address

The LEA instruction loads one of the 16-bit general, index, or pointer registers from another register or memory.

Example:

LEA CX,[SI]

copies the number (address) in the SI register to the CX register.

[Flags affected - none]

LES

Load Extra Segment

The LES instruction performs two distinct operations. First it loads two consecutive bytes of memory into one of the 16-bit general, index, or pointer registers. Then it loads the next two consecutive bytes of memory into the 16-bit ES register.

For example, if DI=1000 then:

LES BX,[DI]

copies the contents of memory locations 1000 and 1001 of the data segment into register BX and the contents of memory locations 1002 and 1003 of the data segment into register ES. [Flags affected - none]

MOV

MOVe

The MOV instruction copies the contents of a register, memory location, or immediate number to a register or memory location. The source and destination must both be of the same length and both cannot be memory locations. [Flags affected - none]

SAHF

Store AH in Flags

The SAHF instruction copies AH to the low-order byte of the flag (status) register. The low-order byte of the 8086/8088 status register is the same as that of the 8085. This instruction is used primarily to translate 8085 software into 8086/8088 software. After this instruction is executed SF, ZF, AF, PF, and CF will correspond to bits 7, 6, 4, 2, and 1 of AH respectively. [Flags affected - SF, ZF, AF, PF, CF]

XCHG

eXCHanGe

The XCHG instruction exchanges the contents of two registers or a register and a memory location. Segment registers cannot be used nor can two memory locations. The source and destination must be of the same length. [Flags affected - none]

XLAT

trans(X)LATe

The XLAT instruction is used to look up values in a table. First the location of the beginning of the table must be loaded into the BX register. Then the relative location within the table of the desired value must be placed in the AL register. When the XLAT instruction is executed the value of BX is added to AL to form an address. The contents of that address then replaces the former value in AL. This instruction can be used to translate ASCII values into EBCDIC values for example. [Flags affected - none]

Flag Instructions

CLC

CLear Carry flag

The CLC instruction places a zero (0) in the carry flag bit of the status register. [Flags affected - CF=0]

CLD

CLear Direction flag (auto-increment)

The CLD instruction places a zero (0) in the direction flag bit of the status register. When this flag is cleared (0), SI and DI will automatically increment when certain string instructions are executed. [Flags affected - DF=0]

CLI

CLear Interrupt-enable flag

The CLI instruction places a zero (0) in the interrupt-enable flag bit of the status register. When this flag is cleared (0) the 8086/8088 will not respond to interrupt signals on the INTR pin. Signals on the NMI pin are not affected however. [Flags affected - IF=0]

CMC

CoMplement Carry flag

The CMC instruction inverts the carry flag bit of the status register. If the CF is 0, it will be changed to a 1. If it is a 1, it will be changed to 0. [Flags affected - CF]

STC

SeT Carry flag

The STC instruction places a one (1) in the carry flag bit of the status register. [Flags affected - CF=1]

STD

SeT Direction flag (auto-decrement)

The STD instruction places a one (1) in the direction flag bit of the status register. When this flag is set (1), SI and DI will automatically decrement when certain string instructions are executed. [Flags affected - DF=1]

STI

SeT Interrupt enable flag

The STI instruction places a one (1) in the interrupt-enable flag bit of the status register. When this flag is set (1) the 8086/8088 will respond to interrupt signals on the INTR pin. [Flags affected - IF=1]

Arithmetic Instructions

AAA

ASCII Adjust for Addition

The AAA instruction can be used after addition to adjust or alter the number in AL to what it would be if the last two operands were ASCII numbers. AH will be cleared. [Flags affected - AF, CF, OF (undefined), SF (undefined), ZF (undefined), PF (undefined)]

AAD

ASCII Adjust for Division

The AAD instruction is used before division by a single-digit, unpacked, BCD number. First you must have an unpacked, two-digit, BCD number in AX. The AAD instruction can then be used to adjust that number. This adjustment must occur **before** any division can take place. The adjustment changes the two-digit, unpacked, BCD number in AX into its equivalent binary number in AL. AH is changed to 00h. Next, AX can be divided by an 8-bit, single-digit, unpacked, BCD number. The binary quotient will be in AL with the binary remainder in AH. **Note: To use this instruction with ASCII numbers the "3" in the upper nibble must be masked out of the numbers first.** [Flags affected - SF, ZF, PF, OF (undefined), AF (undefined), CF (undefined)]

EXPANDED TABLE OF 8086/8088 INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

AAM

ASCII Adjust for Multiplication

The AAM instruction adjusts the product after multiplication of two, unpacked, single-digit, BCD numbers. To use this instruction you must have two single-digit, unpacked, BCD numbers. One must be in AL and the other in a register or memory location. After you multiply the two single-digit, unpacked, BCD numbers the binary answer will be in AL. The AAM instruction will convert it to its unpacked BCD equivalent. **Note: To use this instruction with ASCII numbers you must first mask the "3" in the upper nibble.** [Flags affected - SF, ZF, PF, AF (undefined), OF (undefined), CF (undefined)]

AAS

ASCII Adjust for Subtraction

The AAS instruction can be used after subtraction to adjust or alter the number in AL to what it would be if the last two operands were ASCII numbers. AH will be cleared. [Flags affected - AF, CF, OF (undefined), SF (undefined), ZF (undefined), PF (undefined)]

ADC

AdD with Carry

The ADC instruction works the same as the ADD instruction except that it adds the value in the carry flag (CF) to the sum of the two operands. [Flags affected - CF, PF, AF, ZF, SF, OF]

ADD

ADD

The ADD instruction adds a binary number in a source register, memory location, or immediate number to a destination binary number in a register or memory location. The result is placed in the destination location. The source and destination are assumed to be binary, both must be of the same size (byte or word), and both cannot be memory locations. [Flags affected - CF, PF, AF, ZF, SF, OF]

CBW

Convert Byte to Word

The CBW instruction takes bit 7 (the highest-order bit) of AL and duplicates it in every bit of AH. This converts an 8-bit signed-binary number in AL into a 16-bit signed-binary number in AX. This must be done before division (IDIV) involving two 8-bit **signed-binary** numbers to convert the dividend (in AL) into its 16-bit form (in AX). (For unsigned-binary numbers place 00H in AH.) It can also be used before integer multiplication (IMUL) involving an 8-bit operand and a 16-bit operand. The 8-bit operand can be converted to a 16-bit operand before the IMUL instruction is executed. [Flags affected - none]

CWD

Convert Word to Double word

The CWD instruction is similar to the CBW instruction except that it converts 16-bit values into 32-bit values instead of 8-bit to 16-bit. It takes bit 15 (the highest-order bit) of AX and duplicates it in every bit of DX. This converts a 16-bit signed-binary number in AX into a 32-bit signed-binary number in DX:AX (high 16 bits in DX, low 16 bits in AX). This must be done before division involving two 16-bit numbers to convert the dividend (in AX) into its 32-bit form (in DX:AX). [Flags affected - none]

DAA

Decimal Adjust for Addition

The DAA instruction adjusts the contents of AL from a binary number to a packed BCD (binary coded decimal) number when used after addition. When addition is performed the operands are assumed to be binary numbers. If they were in fact packed BCD numbers then the DAA instruction would have to be used after the addition to correct the result. Note that DAA only works on AL so each byte of a multi-byte packed BCD number must be moved into AL, added, adjusted, and then the result moved back out to make room for the next byte. [Flags affected - SF, ZF, AF, PF, CF, OF (undefined)]

DAS

Decimal Adjust for Subtraction

The DAS instruction adjusts the contents of AL from a binary number to a packed BCD (binary-coded-decimal) number when used after subtraction. When subtraction is performed the operands are assumed to be binary numbers. If they were in fact packed BCD numbers then the DAS instruction would have to be used after the subtraction to correct the result. Note that DAS only works on AL so each byte of a multi-byte packed BCD number must be moved into AL, subtracted, adjusted, and then the result moved back out to make room for the next byte. [Flags affected - SF, ZF, AF, PF, CF, OF (undefined)]

DIV

DIVide (unsigned)

The DIV instruction can divide a 16-bit unsigned-binary number in AX by an 8-bit unsigned-binary number in a register or memory location. If you want to divide one 8-bit number by another you must first change the dividend in AL into a 16-bit number by placing 00H in AH. After execution the result (quotient) will be in AL and the remainder in AH.

DIV can also divide a 32-bit unsigned-binary number in DX:AX (high-order word in DX, low-order word in AX) by a 16-bit unsigned-binary number in a register or memory location. If you wish to divide one 16-bit number by another you must first convert the dividend in AX into a 32-bit number in DX:AX by placing 0000H in DX. The result (quotient) will be in AX and the remainder in DX. [Flags affected - OF (undefined), SF (undefined), ZF (undefined), AF (undefined), PF (undefined), CF (undefined)]

IDIV

Integer **DIV**ision (signed)

The IDIV instruction can divide a 16-bit signed-binary number in AX by an 8-bit signed-binary number in a register or memory location. The result (quotient) will be in AL and the remainder in AH. It can also divide a 32-bit signed-binary number in DX:AX (high-order word in DX, low-order word in AX) by a 16-bit signed-binary number in a register or memory location. The result (quotient) will be in AX and the remainder in DX. **Important! - See CBW and CWD.** [Flags affected - OF (undefined), SF (undefined), ZF (undefined), AF (undefined), PF (undefined), CF (undefined)]

IMUL

Integer **MUL**tiplication (signed)

The IMUL instruction multiplies a signed binary number in a register or memory location times a signed number in AL if 8-bit or AX if 16-bit. If two 8-bit numbers are multiplied then a 16-bit answer will be found in AX. If two 16-bit numbers are multiplied then a 32-bit answer will be found in DX:AX (high byte in DX, low byte in AX). To multiply an 8-bit signed binary number by a 16-bit signed-binary number see the CBW instruction. [Flags affected - OF, CF, SF (undefined), ZF (undefined), AF (undefined), PF (undefined)]

MUL

MULtiply (unsigned)

The MUL instruction multiplies an unsigned binary number in a register or memory location times an unsigned number in AL if 8-bit or AX if 16-bit. If two 8-bit numbers are multiplied then a 16-bit answer will be found in AX. If two 16-bit numbers are multiplied then a 32-bit answer will be found in DX:AX (high byte in DX, low byte in AX). [Flags affected - OF, CF, SF (undefined), ZF (undefined), AF (undefined), PF (undefined)]

SBB

SuBtract with Borrow

The SBB instruction is the same as the SUB instruction except that the value in the carry flag (CF) is also subtracted. That is, the source (second operand) and CF are both subtracted from the destination (first operand). The source and destination must both be either 8-bit or 16-bit. All values are assumed to be binary. [Flags affected - OF, SF, ZF, AF, PF, CF]

SUB

SUBtract

The SUB instruction subtracts the contents of a source (the second operand in 8086/8088 assembly language) register, memory location, or an immediate number from the contents of a destination (the first operand in 8086/8088 assembly language) register or memory location. The result is placed in the destination location. The source and destination must both be of the same size (byte or word) and both cannot be memory locations. [Flags affected - CF, PF, AF, ZF, SF, OF]

Logical Instructions

AND

logical **AND**

The AND instruction performs a logical AND of each bit of the source and destination operands. The source (second operand in 8086/8088 assembly language) can be an immediate number, register, or memory location. The destination can be a register or memory location. Both source and destination cannot be memory locations. Both operands can be 8-bit or both can be 16-bit. Neither can be a segment register. After execution the source is unchanged but the destination will contain the result of the ANDing operation. [Flags affected - OF=0, SF, ZF, PF, CF=0, AF (undefined)]

NEG

NEGate (2's complement)

The NEG instruction produces the 2's complement of a binary number. This can be done manually by inverting each bit then adding one (1). This instruction is also essentially the same as subtracting the number from zero. [Flags affected - OF, SF, ZF, AF, PF, CF]

NOT

NOT

The NOT instruction inverts every bit of the operand. The operand can be in a register or memory location. [Flags affected - none]

OR

OR

The OR instruction performs a logical OR of each bit of the source and destination operands. The source (second operand in 8086/8088 assembly language) can be an immediate number, register, or memory location. The destination can be a register or memory location. Both source and destination cannot be memory locations. Both operands can be 8-bit or both can be 16-bit. Neither can be a segment register. After execution the source is unchanged but the destination will contain the result of the ORing operation. [Flags affected - OF=0, SF, ZF, PF, CF=0, AF (undefined)]

XOR

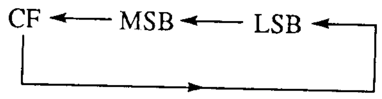
eXclusive **OR**

The XOR instruction performs a logical XOR of each bit of the source and destination operands. The source (second operand in 8086/8088 assembly language) can be an immediate number, register, or memory location. The destination can be a register or memory location. Both source and destination cannot be memory locations. Both operands can be 8-bit or both can be 16-bit. Neither can be a segment register. After execution the source is unchanged but the destination will contain the result of the XORing operation. [Flags affected - OF=0, SF, ZF, PF, CF=0, AF (undefined)]

Rotate and Shift Instructions

RCL

Rotate through Carry to the Left



The RCL instruction rotates the bits of the destination as shown above. After an RCL instruction the destination will have rotated toward the left, the carry flag will hold the bit most recently rotated out of the MSB, and the LSB will hold the bit most recently rotated from the carry flag. The destination can be a register or memory location. If you want to rotate one bit position you specify a "1" in the instruction. If you want to rotate more than one bit position place the number of bits in the CL register and include that register in the instruction.

Examples:

RCL AX,1

rotates AX one bit position

RCL AX,CL

rotates AX the number of bit positions indicated by the number held in the CL register.

[Flags affected - OF, CF]

RCR

Rotate through Carry to the Right

$$\begin{tikzcd} CF \arrow[r] & MSB \arrow[r] & LSB \arrow[ll, bend left] \end{tikzcd}$$

The RCR instruction rotates the bits of the destination as shown above. After an RCR instruction the destination will have rotated toward the right, the carry flag will hold the bit most recently rotated from the LSB, and the MSB will hold the bit most recently rotated from the carry flag. The destination can be a register or memory location. If you want to rotate one bit position you specify a "1" in the instruction. If you want to rotate more than one bit position place the number of bits in the CL register and include that register in the instruction.

Examples:

RCR AX,1

rotates AX one bit position

RCR AX,CL

rotates AX the number of bit positions indicated by the number held in the CL register.

[Flags affected - OF, CF]

ROL

ROtate Left

$$\begin{tikzcd} CF & MSB \arrow[l] \arrow[r, bend right] & LSB \arrow[l] \end{tikzcd}$$

The ROL instruction rotates the bits of the destination as shown above. After an ROL instruction the destination will have rotated toward the left, and the carry flag and the LSB will both contain the same bit which was most recently rotated into them from the MSB. The destination can be a register or memory location. If you want to rotate one bit position you specify a "1" in the instruction. If you want to rotate more than one bit position place the number of bits in the CL register and include that register in the instruction.

Examples:

ROL AX,1

rotates AX one bit position

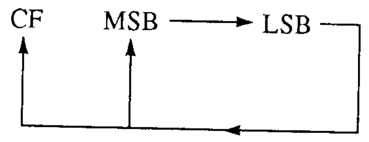
ROL AX,CL

rotates AX the number of bit positions indicated by the number held in the CL register.

[Flags affected - OF, CF]

ROR

ROtate Right



The ROR instruction rotates the bits of the destination as shown above. After an ROR instruction the destination will have rotated toward the right, and the carry flag and the MSB will both contain the same bit which was most recently rotated into them from the LSB. The destination can be a register or memory location. If you want to rotate one bit position you specify a "1" in the instruction. If you want to rotate more than one bit position place the number of bits in the CL register and include that register in the instruction.

Examples:

ROR AX,1

rotates AX one bit position

ROR AX,CL

rotates AX the number of bit positions indicated by the number held in the CL register.

[Flags affected - OF, CF]

SAL/SHL

Shift Arithmetic Left/SHift logical Left

$$\begin{tikzcd} CF & MSB \arrow[l] & LSB \arrow[l] & 0 \arrow[l] \end{tikzcd}$$

The SAL or SHL instruction shifts the bits of the destination as shown above. After an SAL/SHL instruction the destination will have shifted toward the left, the carry flag will contain the bit most recently shifted out of the MSB, and the LSB will contain a 0. The destination can be a register or memory location. If you want to rotate one bit position you specify a "1" in the instruction. If you want to rotate more than one bit position place the number of bits in the CL register and include that register in the instruction.

Examples:

SHL AX,1

rotates AX one bit position

SHL AX,CL

rotates AX the number of bit positions indicated by the number held in the CL register.

(DEBUG Note: DEBUG only allows the SHL mnemonic.)

[Flags affected - OF, SF, ZF, PF, CF, AF (undefined)]

SAR

Shift Arithmetic Right

$$\begin{tikzcd}
MSB \arrow[loop left] \arrow[r] & LSB \arrow[r] & CF
\end{tikzcd}$$

The SAR instruction shifts the bits of the destination as shown above. After an SAR instruction the destination will have shifted to the right, the MSB will contain what it did before the instruction (i.e., it duplicates itself and shifts a copy of itself to the right), and the carry flag will hold the bit most recently shifted out of the LSB. The destination can be a register or memory location. If you want to rotate one bit position you specify a "1" in the instruction. If you want to rotate more than one bit position place the number of bits in the CL register and include that register in the instruction.

Examples:

SAR AX,1	rotates AX one bit position
SAR AX,CL	rotates AX the number of bit positions indicated by the number held in the CL register.

[Flags affected - OF, SF, ZF, PF, CF, AF (undefined)]

SHR

SHift logical **R**ight

$$\begin{tikzcd}
0 \arrow[r] & MSB \arrow[r] & LSB \arrow[r] & CF
\end{tikzcd}$$

The SHR instruction shifts the bits of the destination as shown above. After an SHR instruction the destination will have shifted toward the right, the MSB will contain a 0, and the carry flag will hold the bit most recently shifted in from the LSB. The destination can be a register or memory location. If you want to rotate one bit position you specify a "1" in the instruction. If you want to rotate more than one bit position place the number of bits in the CL register and include that register in the instruction.

Examples:

SHR AX,1	rotates AX one bit position
SHR AX,CL	rotates AX the number of bit positions indicated by the number held in the CL register.

[Flags affected - OF, SF, ZF, PF, CF, AF (undefined)]

Increment and Decrement Instructions

DEC

DECrement

The DEC instruction decreases the value in the destination by 1. The destination is assumed to be a binary number and can be a register (except a segment register) or memory location. It is worthwhile to note that the CF is not affected by this instruction. [Flags affected - OF, SF, ZF, AF, PF]

INC

INCrement

The INC instruction increases the value in the destination by 1. The destination is assumed to be a binary number and can be a register (except a segment register) or memory location. It is worthwhile to note that the CF is not affected by this instruction. [Flags affected - OF, SF, ZF, AF, PF]

Unconditional Jump Instructions

JMP

JuMP

JMP is an unconditional jump instruction which causes the 8086/8088 to continue executing instructions at some other place in the program. The jump can be classified as short, near, or far. The short and near instructions are relative to the current instruction pointer (IP) location. Since the IP always points to the next instruction to be executed you start counting forward or backward from the next instruction after the JMP instruction. A short jump can be up to a maximum of 127 memory bytes forward from the current IP position ($7E_{16}$ or $+127_{10}$) or up to 128 memory bytes backward from the current IP position (80_{16} or -128_{10}). A near jump can be anywhere within the current 64K code segment. The assembler will calculate this as being up to 32,767 bytes forward ($7FFF_{16}$ or $+32{,}767_{10}$) or 32,768 bytes backward (8000_{16} or $-32{,}768_{10}$) from the current IP position. A far jump can be anywhere in the 1-Mbyte addressing range of the 8086/8088. The far jump specifies both the desired code segment (CS) and the desired instruction pointer (IP). **DEBUG Note: When you want to JMP you do not need to be concerned about calculating the distance forward or backward from the current instruction pointer (IP) position. Simply specify the location you want to go to in the form**

JMP XXXX

where XXXX is the memory location (and therefore the desired instruction pointer value) for the short and near jumps and DEBUG will determine whether this is a forward or backward jump and will calculate the exact distance for you. Likewise if you want to use the value in a register as your destination simply specify that register and DEBUG will calculate the relative jump distance for you. In the case of the far jump specify the location you want to jump to in the form

JMP YYYY:XXXX

where YYYY is the code segment (CS) and XXXX is the instruction pointer (IP). [Flags affected - none]

Test (Compare) Instructions

CMP	CoMPare The CMP instruction is used to compare two operands for the purpose of affecting flags according to the outcome. That is, the compare instruction subtracts the source operand (the second operand) from the destination (the first operand). Neither operand is changed; only the flags are affected. The source can be an immediate number, a register, or a memory location. The destination can be a register or memory location. Both operands cannot be memory locations. [Flags affected - OF, SF, ZF, AF, PF, CF]
TEST	**TEST** The TEST instruction ANDs the source and destination operands but neither stores a result nor changes either operand. Rather, the flags are affected by the ANDing. This is useful before a conditional jump instruction. The source can be an immediate number, register, or memory location. The destination can be a register or memory location. Both operands cannot be memory locations. [Flags affected - OF=0, CF=0, SF, ZF, AF (undefined), PF (only lower 8 bits of destination)]

Conditional Jump (Branch) Instructions

JA/JNBE	Jump if Above/Jump if Not Below nor Equal The JA/JNBE conditional jump instruction will cause program execution to transfer to another location in a range from +127 bytes to -128 bytes from the instruction following the jump instruction if CF=0 and if ZF=0 (both must be 0). If this condition is not true no jump occurs. When used after CMP, this instruction is referring to the unsigned values of the operands used by the CMP instruction. **DEBUG Note: Regardless of which mnemonic is used during assembly, DEBUG always disassembles this op code as JA.** [Flags affected - none]
JAE/JNB/JNC	Jump if Above or Equal/Jump if Not Below/Jump if No Carry The JAE/JNB/JNC conditional jump instruction will cause program execution to transfer to another location in a range from +127 bytes to -128 bytes from the instruction following the jump instruction if CF=0. If this condition is not true no jump occurs. When used after CMP, this instruction is referring to the unsigned values of the operands used by the CMP instruction. **DEBUG Note: Regardless of which mnemonic is used during assembly, DEBUG always disassembles this op code as JNB.** [Flags affected - none]
JB/JNAE/JC	Jump if Below/Jump if Not Above nor Equal/Jump if Carry The JB/JNAE/JC conditional jump instruction will cause program execution to transfer to another location in a range from +127 bytes to -128 bytes from the instruction following the jump instruction if CF=1. If this condition is not true no jump occurs. When used after CMP, this instruction is referring to the unsigned values of the operands used by the CMP instruction. **DEBUG Note: Regardless of which mnemonic is used during assembly, DEBUG always disassembles this op code as JB.** [Flags affected - none]

JBE/JNA	Jump if Below or Equal/Jump if Not Above The JBE/JNA conditional jump instruction will cause program execution to transfer to another location in a range from +127 bytes to -128 bytes from the instruction following the jump instruction if CF=0 or ZF=1. If this condition is not true no jump occurs. When used after CMP, this instruction is referring to the unsigned values of the operands used by the CMP instruction. **DEBUG Note: Regardless of which mnemonic is used during assembly, DEBUG always disassembles this op code as JBE.** [Flags affected - none]
JCXZ	Jump if CX register is Zero The JCXZ conditional jump instruction will cause program execution to transfer to another location in a range from +127 bytes to -128 bytes from the instruction following the jump instruction if the CX register is 0. If this condition is not true no jump occurs. [Flags affected - none]
JE/JZ	Jump if Equal to/Jump if Zero The JE/JZ conditional jump instruction will cause program execution to transfer to another location in a range from +127 bytes to -128 bytes from the instruction following the jump instruction if ZF=1. If this condition is not true no jump occurs. When used after CMP, this instruction is referring to the values of the operands used by the CMP instruction. **DEBUG Note: Regardless of which mnemonic is used during assembly, DEBUG always disassembles this op code as JZ.** [Flags affected - none]
JG/JNLE	Jump if Greater/Jump if Not Less than nor Equal The JG/JNLE conditional jump instruction will cause program execution to transfer to another location in a range from +127 bytes to -128 bytes from the instruction following the jump instruction if (SF XOR OF) OR ZF = 0. To say it another way, the jump occurs if the sign flag and the overflow flag are equal (both 0 or both 1) at the same time that the zero flag is 0. Only two combinations are possible. If SF=0, OF=0, and ZF=0 the jump occurs; or if SF=1, OF=1, and ZF=0 the jump also occurs. If this condition is not true no jump occurs. When used after CMP, this instruction is referring to the signed values of the operands used by the CMP instruction. **DEBUG Note: Regardless of which mnemonic is used during assembly, DEBUG always disassembles this op code as JG.** [Flags affected - none]
JGE/JNL	Jump if Greater than or Equal/Jump if Not Less The JGE/JNL conditional jump instruction will cause program execution to transfer to another location in a range from +127 bytes to -128 bytes from the instruction following the jump instruction if SF=OF. If this condition is not true no jump occurs. When used after CMP, this instruction is referring to the signed values of the operands used by the CMP instruction. **DEBUG Note: Regardless of which mnemonic is used during assembly, DEBUG always disassembles this op code as JGE.** [Flags affected - none]

JL/JNGE	Jump if Less/Jump if Not Greater than nor Equal The JGE/JNL conditional jump instruction will cause program execution to transfer to another location in a range from +127 bytes to -128 bytes from the instruction following the jump instruction if the SF does **not** equal the OF. If this condition is not true no jump occurs. When used after CMP, this instruction is referring to the signed values of the operands used by the CMP instruction. **DEBUG Note: Regardless of which mnemonic is used during assembly, DEBUG always disassembles this op code as JL.** [Flags affected - none]
JLE/JNG	Jump if Less than or Equal/Jump if Not Greater The JLE/JNG conditional jump instruction will cause program execution to transfer to another location in a range from +127 bytes to -128 bytes from the instruction following the jump instruction if (SF XOR OF) OR ZF = 1. To say it another way, the jump occurs if the sign flag and the overflow flag are **not** equal, **or** if the zero flag is 0. Only two combinations do **not** produce the jump. If SF=0, OF=0, and ZF=0 then **no** jump occurs; or if SF=1, OF=1, and ZF=0 then **no** jump occurs. When used after CMP, this instruction is referring to the signed values of the operands used by the CMP instruction. **DEBUG Note: Regardless of which mnemonic is used during assembly, DEBUG always disassembles this op code as JLE.** [Flags affected - none]
JNE/JNZ	Jump if Not Equal to/Jump if Not Zero The JNE/JNZ conditional jump instruction will cause program execution to transfer to another location in a range from +127 bytes to -128 bytes from the instruction following the jump instruction if ZF=0. If this condition is not true no jump occurs. When used after CMP, this instruction is referring to the values of the operands used by the CMP instruction. **DEBUG Note: Regardless of which mnemonic is used during assembly, DEBUG always disassembles this op code as JNZ.** [Flags affected - none]
JNO	Jump if Not Overflow An overflow occurs when the result of a signed arithmetic operation is too large to fit in the register or memory location. The JNO conditional jump instruction will cause program execution to transfer to another location in a range from +127 bytes to -128 bytes from the instruction following the jump instruction if OF=0, that is, if an overflow has **not** occurred. If this condition is not true no jump occurs. [Flags affected - none]
JNP/JPO	Jump if Not Parity/Jump if Parity Odd When the result of an operation which affects the parity flag has a result which has an odd number of 1s in it then the PF=0. The JNP/JPO conditional jump instruction will cause program execution to transfer to another location in a range from +127 bytes to -128 bytes from the instruction following the jump instruction if PF=0. If this condition is not true no jump occurs. **DEBUG Note: Regardless of which mnemonic is used during assembly, DEBUG always disassembles this op code as JPO.** [Flags affected - none]

JNS

Jump if Not Sign

The JNS conditional jump instruction will cause program execution to transfer to another location in a range from +127 bytes to -128 bytes from the instruction following the jump instruction if SF=0. If this condition is not true no jump occurs. Since a "0" in the sign flag occurs when the result of the last operation was a positive signed number, this instruction is essentially saying to jump if the last operation produced a positive signed result. [Flags affected - none]

JO

Jump if Overflow

An overflow occurs when the result of a signed arithmetic operation is too large to fit in the register or memory location. The JO conditional jump instruction will cause program execution to transfer to another location in a range from +127 bytes to -128 bytes from the instruction following the jump instruction if OF=1, that is, if an overflow has occurred. If this condition is not true no jump occurs. [Flags affected - none]

JP/JPE

Jump if Parity/Jump if Parity Even

When the result of an operation which affects the parity flag has a result which has an even number of 1s in it then the PF=1. The JP/JPE conditional jump instruction will cause program execution to transfer to another location in a range from +127 bytes to -128 bytes from the instruction following the jump instruction if PF=1. If this condition is not true no jump occurs. **DEBUG Note: Regardless of which mnemonic is used during assembly, DEBUG always disassembles this op code as JPE.** [Flags affected - none]

JS

Jump if Sign

The JS conditional jump instruction will cause program execution to transfer to another location in a range from +127 bytes to -128 bytes from the instruction following the jump instruction if SF=1. If this condition is not true no jump occurs. Since a "1" in the sign flag occurs when the result of the last operation was a negative signed number, this instruction is essentially saying to jump if the last operation produced a negative signed result. [Flags affected - none]

Subroutine Instructions

CALL

CALL procedure

The CALL instruction causes the 8086/8088 to leave its current location in the program and to begin executing a procedure (a small special purpose program or subroutine located in a different place in memory) and then automatically return after that procedure is finished. The call can be classified as near or far. The near instruction is relative to the current instruction pointer (IP) location. Since the IP always points to the next instruction to be executed you start counting forward or backward from the next instruction after the CALL instruction. A near call can be anywhere within the current 64K code segment. The assembler will calculate this as being up to 32,767 bytes forward ($7FFF_{16}$ or $+32{,}767_{10}$) or 32,768 bytes backward (8000_{16} or $-32{,}768_{10}$) from the current IP position. When a near call is executed the contents of the instruction pointer (IP) are pushed onto the stack so that the 8086/8088 will know where to return after the procedure has been finished. A far call can be anywhere in the 1-Mbyte addressing range of the 8086/8088. The far call specifies both the desired code segment (CS) and the desired instruction pointer

(IP). When a far call is executed the contents of both the instruction pointer (IP) and the code segment (CS) are pushed onto the stack so that the 8086/8088 will know where to return after the procedure has been finished. **DEBUG Note: When you want to CALL a procedure you do not need to be concerned about calculating the distance forward or backward from the current instruction pointer (IP) position. Simply specify the location of the procedure in the form**

CALL XXXX

where XXXX is the memory location (and therefore the desired instruction pointer value) for the near call and DEBUG will determine whether that location is forward or backward and will calculate the exact distance for you. Likewise if you want to use the value in a register as your destination simply specify that register and DEBUG will calculate the relative distance for you. In the case of a far call specify the location of the procedure in the form

CALL YYYY:XXXX

where YYYY is the code segment (CS) and XXXX is the instruction pointer (IP). (See also RETurn.) [Flags affected - none]

RET

RETurn from subroutine

The RET instruction is placed at the end of a procedure or subroutine. It marks the end of that procedure and causes the 8086/8088 to return to the instruction immediately following the CALL instruction which began this particular procedure. The 8086/8088 knows where to return because the CALL instruction pushed the contents of the instruction pointer (IP) onto the stack. The RET instruction pops the value of the IP from the stack and places it in the IP. In the case of a far call the return instruction pops both the IP value and the code segment (CS) value from the stack. **DEBUG Note: DEBUG accepts both RET and RETN as the mnemonics for a return from a near call. When disassembled both will appear as RET. To specify a return from a far call the mnemonic RETF must be used and it will be disassembled as RETF.** [Flags affected - none]

Stack Instructions

POP

POP from stack

The POP instruction copies the word at the top of the stack to the destination operand. The destination can be a general-purpose register, segment register, or two consecutive memory locations. (The CS register is illegal.) After the POP, the stack pointer (SP) is incremented by 2 to point to the new top-of-stack. [Flags affected - none]

POPF

POP Flags from stack

The POPF instruction copies the word at the top of the stack into the flag register, replacing the values of all flags. The stack pointer (SP) is then incremented by 2. (Using POPF and PUSHF provides a way to change the TF. There is no instruction for directly altering this flag.) [Flags affected - OF, DF, IF, TF, SF, ZF, AF, PF, CF]

PUSH

PUSH onto stack

The PUSH instruction decrements the stack pointer (SP) by 2 and then copies the source operand (word) to the new top-of-stack. The source can be a general-purpose register, segment register, or two consecutive memory locations. [Flags affected - none]

PUSHF

PUSH Flags onto stack

The PUSHF instruction decrements the stack pointer (SP) by 2 and then copies the flag register to the new top-of-stack. [Flags affected - none]

Interrupt Instructions

INT

INTerrupt

The INT instruction causes program execution to be transferred to a special type of routine whose address is pointed to by an interrupt vector. There are 256 interrupt vectors in memory locations 00000h to 003FFh. Each vector is 4 bytes in length and contains the address (CS:IP) of the routine which handles this particular type of interrupt. The INT operand is a decimal number from 0 through 255 which identifies which interrupt vector is to be used. The actual memory location of the interrupt is calculated by multiplying the operand by 4. That answer forms the decimal equivalent of the beginning of the four memory locations which hold the interrupt vector. When the INT instruction is executed the following occur:

1. The stack pointer is decremented by 2 and the flags are pushed onto the stack.
2. IF and TF are cleared.
3. The stack pointer is decremented by 2 and CS is pushed onto the stack.
4. The new CS is fetched from the interrupt vector and the interrupt vector + 1.
5. The stack pointer is decremented by 2 and IP is pushed onto the stack.
6. The new IP is fetched from the interrupt vector + 2 and the interrupt vector + 3.
7. Begin execution of the interrupt routine located at memory location CS:IP.

The routine will continue until a IRET instruction is encountered, at which point program execution will pick up where it left off immediately after the INT instruction. [Flags affected - IF and TF]

INTO

INTerrupt on **O**verflow

The INTO instruction initiates a software interrupt which is, in all respects, the same as that produced by the INT instruction except that the INTO instruction is conditional, and the operand cannot be specified but is automatically type 4. That is, the INTO instruction will branch to the interrupt routine only if OF=1 and there is no choice as to where the interrupt vector will come from. It will always be a type 4 interrupt which is held in the 4 bytes starting at memory location 10h. This instruction is most often used after arithmetic operations to handle any overflow conditions. See the discussion for the INT instruction for more details. [Flags affected - IF and TF]

IRET

Interrupt **RET**urn

The IRET instruction is used to return from an interrupt routine (whether a hardware or software interrupt). The IP, CS, and flags are all popped from the stack and program execution continues from the instruction immediately following the INT instruction. The IRET instruction has no operand. [Flags affected - OF, DF, IF, TF, SF, ZF, AF, PF, CF]

Input-Output Instructions

IN

INput

The IN instruction allows a byte or word to be acquired from an I/O device [source] and placed in AL (byte) or AX (word) [destination]. An I/O address [source operand] from 00h through FFh can be specified directly in the instruction. If an address larger than FFh is desired a 16-bit address can be placed in DX used as the source operand in the IN instruction. Only AX and AL can be used as destinations [destination operand] by the IN instruction.

Example:

IN AL,45	copy a byte from I/O address 45h into AL
IN AX,78	copy a word from I/O address 78h into AX
IN AL,DX	copy a byte from the I/O address pointed to by the contents of DX and place in AL

I/O port addresses F8h through FFh are reserved by Intel for future hardware and software products and should not be used for any other purpose. [Flags affected - none]

OUT

OUTput

The OUT instruction allows a byte or word to be sent from AL (byte) or AX (word) [source] to an I/O device [destination]. An I/O address [destination operand] from 00h through FFh can be specified directly in the instruction. If an address larger than FFh is desired a 16-bit address can be placed in DX used as the destination operand in the OUT instruction. Only AX and AL can be used as sources [source operand] by the OUT instruction.

Example:

OUT 45,AL	copy a byte from AL to I/O address 45h

OUT 78,AX	copy a word from AX to I/O address 78h
OUT DX,AL	copy a byte from AL to the I/O address pointed to by the contents of DX

I/O port addresses F8h through FFh are reserved by Intel for future hardware and software products and should not be used for any other purpose. [Flags affected - none]

String Instructions

CMPS/CMPSB/CMPSW CoMpare Strings/CoMPare Strings Byte/CoMPare Strings Word

The CMPS/CMPSB/CMPSW instruction is used to compare the contents of two memory bytes, two words, or two entire sections of memory. The SI (source index) is used to point to the source in the DS (data segment). The DI (destination index) is used to point to the destination in the ES (extra segment). The 8086/8088 makes the comparison by subtracting the destination from the source. Neither operand is changed by the comparison; only flags are affected. After the comparison DI and SI are automatically incremented (if DF=0) or decremented (if DF=1). The increment/decrement is 1 if the CMPB mnemonic is used or 2 if CMPW is used. The REP/REPE/REPZ and REPNE/REPNZ repeat prefixes can be used with this instruction to compare an entire section of memory. **DEBUG Note: Only the CMPSB and CMPSW mnemonics are accepted by DEBUG.** [Flags affected - OF, SF, ZF, AF, PF, CF]

LODS/LODSB/LODSW LOaD String/LOaD String Byte/LOaD String Word

The LODS/LODSB/LODSW instruction loads (copies) either a byte (LODSB) from the memory location pointed to by SI into AL, or a word (LODSW) from the memory location pointed to by SI into AX. SI is either automatically incremented by 1 (LODSB) or by 2 (LODSW) if DF=0, or SI is automatically decremented by 1 (LODSB) or by 2 (LODSW) if DF=1. The REP/REPE/REPZ and REPNE/REPNZ repeat prefixes can be used with this instruction. **DEBUG Note: DEBUG only accepts the LODSB and LODSW mnemonics.** [Flags affected - none].

MOVS/MOVSB/MOVSW MOVe String/MOVe String Byte/MOVe String Word

The MOVS/MOVSB/MOVSW instruction is used to transfer the contents of a block of memory to another area in memory. The SI (source index) is used to point to the source in the DS (data segment). The DI (destination index) is used to point to the destination in the ES (extra segment). After the move DI and SI are automatically incremented (if DF=0) or decremented (if DF=1). The increment/decrement is 1 if the MOVSB mnemonic is used or 2 if MOVSW is used. The REP/REPE/REPZ and REPNE/REPNZ repeat prefixes can be used with this instruction to move an entire section of memory. **DEBUG Note: Only the MOVSB and MOVSW mnemonics are accepted by DEBUG.** [Flags affected - none]

REP/REPE/REPZ	**REP**eat/**REP**eat if **E**qual/**REP**eat if **Z**ero REP/REPE/REPZ is a prefix which causes string instructions to be repeated the number of times indicated by the value in CX. Each time the string instruction is repeated CX is decremented by one. This continues 1) in the case of MOVS and STOS, **until** CX=0, or 2) in the case of CMPS and SCAS, **until either** CX=0 **or** the compared bytes or words are **not** equal (ie. ZF=0). **DEBUG Note: REP, REPE, and REPZ are all mnemonics for the same op code and DEBUG disassembles all of them as REPZ.** [Flags affected - none]
REPNE/REPNZ	**REP**eat if **N**ot **E**qual/**REP**eat if **N**ot **Z**ero REPNE/REPNZ is a prefix which causes string instructions to be repeated the number of times indicated by the value in CX. Each time the string instruction is repeated CX is decremented by 1. This continues 1) in the case of MOVS and STOS, **until** CX=0, or 2) in the case of CMPS and SCAS, **until either** CX=0 **or** the compared bytes or words **are** equal (ie. ZF=1). **DEBUG Note: REPNE and REPNZ are mnemonics for the same op code and DEBUG disassembles all of them as REPNZ.** [Flags affected - none]
SCAS/SCASB/SCASW	**SCA**n **S**tring/**SCA**n **S**tring **B**yte/**SCA**n **S**tring **W**ord The SCAS/SCASB/SCASW instruction is used to check a string for the occurrence or non-occurrence of a particular byte or word. The instruction accomplishes this by subtracting the byte or word in the extra segment (ES) which is pointed to by DI from AL (if a byte) or AX (if a word). Neither the contents of the string nor those of AX/AL are changed; however the flags are affected by the operation. After the operation, DI is automatically incremented (if DF=0) or decremented (if DF=1). DI will be incremented or decremented by 1 for byte scans or by 2 for word scans. The REP/REPE/REPZ prefix can be used to scan for the non-occurrence of a byte or word. The REPNE/REPNZ prefix can be used to scan for the occurrence of a byte or word. **DEBUG Note: DEBUG only recognizes the SCASB and SCASW mnemonics.** [Flags affected - OF, SF, ZF, AF, PF, CF]
STOS/STOSB/STOSW	**STO**re **S**tring/**STO**re **S**tring **B**yte/**STO**re **S**tring **W**ord The STOS/STOSB/STOSW instruction copies a byte from AL or a word from AX to a memory location in the extra segment (ES) pointed to by DI. After the operation, DI is automatically incremented (if DF=0) or decremented (if DF=1). DI will be incremented or decremented by 1 for a byte store or by 2 for a word store. The REP/REPE/REPZ and REPNE/REPNZ repeat prefixes can be used with this instruction to store a certain value in a range of memory locations. **DEBUG Note: Only the STOSB and STOSW mnemonics are accepted by DEBUG.** [Flags affected - none]

Loop Instructions

LOOP	**LOOP** The LOOP instruction provides a way to repeat a group of instructions the number of times indicated by the value in the CX register. The LOOP instruction unconditionally transfers program execution to a memory location in the range of -128 to +127 bytes from the address of the instruction immediately following the

LOOP instruction if CX > 0. Each time the LOOP instruction is executed CX is decremented by 1; then the value of CX is checked. If CX > 0, program execution will branch to the location indicated by the operand of the LOOP instruction. If CX = 0, the program does not branch and the instruction immediately following the LOOP instruction is executed next. As CX is decremented wraparound occurs from 0000h to FFFFh. [Flags affected - none]

LOOPE/LOOPZ

LOOP while **E**qual/**LOOP** while **Z**ero

The LOOPE/LOOPZ instruction provides a way to repeat a group of instructions the number of times indicated by the value in the CX register. The LOOPE/LOOPZ instruction transfers program execution to a memory location in the range of -128 to +127 bytes from the address of the instruction immediately following the LOOP instruction if CX > 0 **and** ZF=1. Each time the LOOP instruction is executed CX is decremented by 1; then the values of CX and ZF are checked. If CX > 0, program execution will branch to the location indicated by the operand of the LOOP instruction if ZF=1 also. If **either** CX = 0 **or** ZF=0, the program does not branch, and the instruction immediately following the LOOP instruction is executed next. As CX is decremented wraparound occurs from 0000h to FFFFh. [Flags affected - none]

LOOPNE/LOOPNZ

LOOP while Not **E**qual/**LOOP** while Not **Z**ero

The LOOPNE/LOOPNZ instruction provides a way to repeat a group of instructions the number of times indicated by the value in the CX register. The LOOPNE/LOOPNZ instruction transfers program execution to a memory location in the range of -128 to +127 bytes from the address of the instruction immediately following the LOOP instruction if CX > 0 **and** ZF=0. Each time the LOOP instruction is executed CX is decremented by 1; then the values of CX and ZF are checked. If CX > 0, program execution will branch to the location indicated by the operand of the LOOP instruction if ZF=0 also. If **either** CX = 0 **or** ZF=1, the program does not branch, and the instruction immediately following the LOOP instruction is executed next. As CX is decremented wraparound occurs from 0000h to FFFFh. [Flags affected - none]

CONDENSED TABLE OF 8086/8088 INSTRUCTIONS LISTED BY CATEGORY

CPU Control Instructions

ESC	**ESC**ape
HLT	**H**a**LT**
LOCK	**LOCK**
NOP	**N**o **OP**eration
WAIT	**WAIT**

CONDENSED TABLE OF 8086/8088 INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Data Transfer Instructions

LAHF	Load AH from Flag
LDS	Load Data Segment
LEA	Load Effective Address
LES	Load Extra Segment
MOV	MOVe
SAHF	Store AH in Flags
XCHG	eXCHanGe
XLAT	trans(X)LATe

Flag Instructions

CLC	CLear Carry flag
CLD	CLear Direction flag (auto-increment)
CLI	CLear Interrupt-enable flag
CMC	CoMplement Carry flag
STC	SeT Carry flag
STD	SeT Direction flag (auto-decrement)
STI	SeT Interrupt enable flag

Arithmetic Instructions

AAA	ASCII Adjust for Addition
AAD	ASCII Adjust for Division
AAM	ASCII Adjust for Multiplication
AAS	ASCII Adjust for Subtraction
ADC	AdD with Carry
ADD	ADD
CBW	Convert Byte to Word
CWD	Convert Word to Double word
DAA	Decimal Adjust for Addition
DAS	Decimal Adjust for Subtraction
DIV	DIVide (unsigned)
IDIV	Integer DIVision (signed)
IMUL	Integer MULtiplication (signed)
MUL	MULtiply (unsigned)
SBB	SuBtract with Borrow
SUB	SUBtract

Logical Instructions

AND	logical AND
NEG	NEGate (2's complement)
NOT	NOT
OR	OR
XOR	eXclusive OR

Rotate and Shift Instructions

RCL	Rotate through Carry to the Left
RCR	Rotate through Carry to the Right
ROL	ROtate Left
ROR	ROtate Right
SAL/SHL	Shift Arithmetic Left/SHift logical LefT
SAR	Shift Arithmetic Right
SHR	SHift logical Right

Increment and Decrement Instructions

DEC	DECrement
INC	INCrement

Unconditional Jump Instructions

JMP	JuMP

Test (Compare) Instructions

CMP	CoMPare
TEST	TEST

Conditional Jump (Branch) Instructions

JA/JNBE	Jump if Above/Jump if Not Below nor Equal
JAE/JNB/JNC	Jump if Above or Equal/Jump if Not Below/Jump if No Carry
JB/JNAE/JC	Jump if Below/Jump if Not Above nor Equal/Jump if Carry
JBE/JNA	Jump if Below or Equal/Jump if Not Above
JCXZ	Jump if CX register is Zero
JE/JZ	Jump if Equal to/Jump if Zero
JG/JNLE	Jump if Greater/Jump if Not Less than nor Equal
JGE/JNL	Jump if Greater than or Equal/Jump if Not Less
JL/JNGE	Jump if Less/Jump if Not Greater than nor Equal
JLE/JNG	Jump if Less than or Equal/Jump if Not Greater
JNE/JNZ	Jump if Not Equal to/Jump if Not Zero
JNO	Jump if Not Overflow
JNP/JPO	Jump if Not Parity/Jump if Parity Odd
JNS	Jump if Not Sign
JO	Jump if Overflow
JP/JPE	Jump if Parity/Jump if Parity Even
JS	Jump if Sign

CONDENSED TABLE OF 8086/8088 INSTRUCTIONS LISTED BY CATEGORY (*Continued*)

Subroutine Instructions

CALL	**CALL** procedure
RET	**RETurn** from subroutine

Stack Instructions

POP	**POP** from stack
POPF	**POP** Flags from stack
PUSH	**PUSH** onto stack
PUSHF	**PUSH** Flags onto stack

Interrupt Instructions

INT	**INTerrupt**
INTO	**INTerrupt** on Overflow
IRET	Interrupt **RETurn**

Input-Output Instructions

IN	**IN**put
OUT	**OUT**put

String Instructions

CMPS/CMPSB/CMPSW	CoMpare Strings/CoMPare Strings Byte/CoMPare Strings Word
LODS/LODSB/LODSW	LOaD String/LOaD String Byte/LOaD String Word
MOVS/MOVSB/MOVSW	MOVe String/MOVe String Byte/MOVe String Word
REP/REPE/REPZ	REPeat/REPeat if Equal/REPeat if Zero
REPNE/REPNZ	REPeat if Not Equal/REPeat if Not Zero
SCAS/SCASB/SCASW	SCAn String/SCAn String Byte/SCAn String Word
STOS/STOSB/STOSW	STOre String/STOre String Byte/STOre String Word

Loop Instructions

LOOP	**LOOP**
LOOPE/LOOPZ	**LOOP** while Equal/**LOOP** while Zero
LOOPNE/LOOPNZ	**LOOP** while Not Equal/**LOOP** while Not Zero

CONDENSED TABLE OF 8086/8088 INSTRUCTIONS LISTED ALPHABETICALLY

AAA	ASCII Adjust for Addition
AAD	ASCII Adjust for Division
AAM	ASCII Adjust for Multiplication
AAS	ASCII Adjust for Subtraction
ADC	AdD with Carry
ADD	ADD
AND	logical AND
CALL	CALL procedure
CBW	Convert Byte to Word
CLC	CLear Carry flag
CLD	CLear Direction flag (auto-increment)
CLI	CLear Interrupt-enable flag
CMC	CoMplement Carry flag
CMP	CoMPare
CMPS/CMPSB/CMPSW	CoMpare Strings/CoMPare Strings Byte/CoMPare Strings Word
CWD	Convert Word to Double word
DAA	Decimal Adjust for Addition
DAS	Decimal Adjust for Subtraction
DEC	DECrement
DIV	DIVide (unsigned)
ESC	ESCape
HLT	HaLT
IDIV	Integer DIVision (signed)
IMUL	Integer MULtiplication (signed)
IN	INput
INC	INCrement
INT	INTerrupt
INTO	INTerrupt on Overflow
IRET	Interrupt RETurn
JA/JNBE	Jump if Above/Jump if Not Below nor Equal
JAE/JNB/JNC	Jump if Above or Equal/Jump if Not Below/Jump if No Carry
JB/JNAE/JC	Jump if Below/Jump if Not Above nor Equal/Jump if Carry
JBE/JNA	Jump if Below or Equal/Jump if Not Above
JCXZ	Jump if CX register is Zero
JE/JZ	Jump if Equal to/Jump if Zero
JG/JNLE	Jump if Greater/Jump if Not Less than nor Equal
JGE/JNL	Jump if Greater than or Equal/Jump if Not Less
JL/JNGE	Jump if Less/Jump if Not Greater than nor Equal
JLE/JNG	Jump if Less than or Equal/Jump if Not Greater
JMP	JuMP unconditional
JNE/JNZ	Jump if Not Equal to/Jump if Not Zero
JNO	Jump if Not Overflow
JNP/JPO	Jump if Not Parity/Jump if Parity Odd
JNS	Jump if Not Sign
JO	Jump if Overflow
JP/JPE	Jump if Parity/Jump if Parity Even
JS	Jump if Sign
LAHF	Load AH from Flag
LDS	Load Data Segment

CONDENSED TABLE OF 8086/8088 INSTRUCTIONS LISTED ALPHABETICALLY *(Continued)*

LEA	Load Effective Address
LES	Load Extra Segment
LOCK	LOCK
LODS/LODSB/LODSW	LOaD String/LOaD String Byte/LOaD String Word
LOOP	LOOP
LOOPE/LOOPZ	LOOP while Equal/LOOP while Zero
LOOPNE/LOOPNZ	LOOP while Not Equal/LOOP while Not Zero
MOV	MOVe
MOVS/MOVSB/MOVSW	MOVe String/MOVe String Byte/MOVe String Word
MUL	MULtiply (unsigned)
NEG	NEGate (2's complement)
NOP	No OPeration
NOT	NOT
OR	OR
OUT	OUTput
POP	POP from stack
POPF	POP Flags from stack
PUSH	PUSH onto stack
PUSHF	PUSH Flags onto stack
RCL	Rotate through Carry to the Left
RCR	Rotate through Carry to the Right
REP/REPE/REPZ	REPeat/REPeat if Equal/REPeat if Zero
REPNE/REPNZ	REPeat if Not Equal/REPeat if Not Zero
RET	RETurn from subroutine
ROL	ROtate Left
ROR	ROtate Right
SAHF	Store AH in Flags
SAL/SHL	Shift Arithmetic Left/SHift logical Left
SAR	Shift Arithmetic Right
SBB	SuBtract with Borrow
SCAS/SCASB/SCASW	SCAn String/SCAn String Byte/SCAn String Word
SHR	SHift logical Right
STC	SeT Carry flag
STD	SeT Direction flag (auto-decrement)
STI	SeT Interrupt enable flag
STOS/STOSB/STOSW	STOre String/STOre String Byte/STOre String Word
SUB	SUBtract
TEST	TEST
WAIT	WAIT
XCHG	eXCHanGe (source with destination)
XLAT	trans(X)LATe
XOR	eXclusive OR

EXPANDED TABLE OF 6502 INSTRUCTIONS LISTED BY CATEGORY

Mne-monic	Operation	Boolean/Arith Operation	Flags NV-BDIZC	Address Mode	Assembler Notation	Op	~	#	Notes
				CPU Control Instructions					
NOP	No OPeration	Nothing	xx-xxxxx	Implied	NOP	EA	2	1	
BRK	BReaK (forced interrupt)	PC + 2 → S SP - 2 → SP PSR → S SP - 1 → S $FFFE → PC	xx-1x1xx	Implied	BRK	00	7	1	
				Data Transfer Instructions					
LDA	LoaD Accumulator	M → A	Nx-xxxZx	Immediate	LDA #$dd	A9	2	2	
				Absolute	LDA $aaaa	AD	4	3	
				Zero Page	LDA $aa	A5	3	2	
				Indxd Indct	LDA ($ff,X)	A1	6	2	
				Indct Indxd	LDA ($aa),Y	B1	5*	2	
				Zero page,X	LDA $ff,X	B5	4	2	
				Absolute,X	LDA $ffff,X	BD	4*	3	
				Absolute,Y	LDA $ffff,Y	B9	4*	3	
LDX	LoaD X register	M → X	Nx-xxxZx	Immediate	LDX #$dd	A2	2	2	
				Absolute	LDX $aaaa	AE	4	3	
				Zero page	LDX $aa	A6	3	2	
				Absolute,Y	LDX $ffff,Y	BE	4*	3	
				Zero page,Y	LDX $ff,Y	B6	4	2	
LDY	LoaD Y register	M → Y	Nx-xxxZx	Immediate	LDY #$dd	A0	2	2	
				Absolute	LDY $aaaa	AC	4	3	
				Zero page	LDY $aa	A4	3	2	
				Zero page,X	LDY $ff,X	B4	4	2	
				Absolute,X	LDY $ffff,X	BC	4*	3	
STA	STore Accumulator	A → M	xx-xxxxx	Absolute	STA $aaaa	8D	4	3	
				Zero page	STA $aa	85	3	2	
				Indxd Indct	STA ($ff,X)	81	6	2	
				Indct Indxd	STA ($aa),Y	91	6	2	
				Zero page,X	STA $ff,X	95	4	2	
				Absolute,X	STA $ffff,X	9D	5	3	
				Absolute,Y	STA $ffff,Y	99	5	3	
STX	STore X register	X → M	xx-xxxxx	Absolute	STX $aaaa	8E	4	3	
				Zero page	STX $aa	86	3	2	
				Zero page,Y	STX $ff,Y	96	4	2	
STY	STore Y register	Y → M	xx-xxxxx	Absolute	STY $aaaa	8C	4	3	
				Zero page	STY $aa	84	3	2	
				Zero page,X	STY $ff,X	94	4	2	
TAX	Transfer Accumulator to X register	A → X	Nx-xxxZx	Implied	TAX	AA	2	1	
TXA	Transfer X register to Accumulator	X → A	Nx-xxxZx	Implied	TXA	8A	2	1	

EXPANDED TABLE OF 6502 INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Mne-monic	Operation	Boolean/Arith Operation	Flags NV-BDIZC	Address Mode	Assembler Notation	Op	~	#	Notes
TAY	Transfer Accumulator to Y register	A → Y	Nx-xxxZx	Implied	TAY	A8	2	1	
TYA	Transfer Y register to Accumulator	Y → A	Nx-xxxZx	Implied	TYA	98	2	1	

Flag Instructions

Mne-monic	Operation	Boolean/Arith Operation	Flags NV-BDIZC	Address Mode	Assembler Notation	Op	~	#	Notes
CLC	CLear Carry flag	0 → C	xx-xxxx0	Implied	CLC	18	2	1	
CLD	CLear Decimal flag	0 → D	xx-x0xxx	Implied	CLD	D8	2	1	
CLI	CLear Interrupt flag	0 → I	xx-xx0xx	Implied	CLI	58	2	1	
CLV	CLear oVerflow flag	0 → V	x0-xxxxx	Implied	CLV	B8	2	1	
SEC	SEt Carry flag	1 → C	xx-xxxx1	Implied	SEC	38	2	1	
SED	SEt Decimal flag	1 → D	xx-x1xxx	Implied	SED	F8	2	1	
SEI	SEt Interrupt flag	1 → I	xx-xx1xx	Implied	SEI	78	2	1	

Arithmetic Instructions

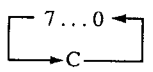
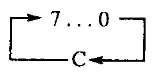
Mne-monic	Operation	Boolean/Arith Operation	Flags NV-BDIZC	Address Mode	Assembler Notation	Op	~	#	Notes
ADC	AdD with Carry	A + M + C → A	NV-xxxZC	Immediate	ADC #$dd	69	2	2	The carry flag must be cleared before single-precision addition or before the first byte of multiple-precision addition.
				Absolute	ADC $aaaa	6D	4	3	
				Zero page	ADC $aa	65	3	2	
				Indxd Indct	ADC ($ff,X)	61	6	2	
				Indct Indxd	ADC ($aa),Y	71	5*	2	
				Zero page,X	ADC $ff,X	75	4	2	
				Absolute,X	ADC $ffff,X	7D	4*	3	
				Absolute,Y	ADC $ffff,Y	79	4*	3	
SBC	SuBtract with Carry	A - M - (1-C) → A	NV-xxxZC	Immediate	SBC #$dd	E9	2	2	The carry flag must be set before single-precision subtraction or before the first byte of multiple-precision subtraction.
				Absolute	SBC $aaaa	ED	4	3	
				Zero page	SBC $aa	E5	3	2	
		Note: (1-C) = Borrow		Indxd Indct	SBC ($ff,X)	E1	6	2	
				Indct Indxd	SBC ($aa),Y	F1	5*	2	
				Zero page,X	SBC $ff,X	F5	4	2	
				Absolute,X	SBC $ffff,X	FD	4*	3	The operation of the carry flag is inverted during subtraction.
				Absolute,Y	SBC $ffff,Y	F9	4*	3	

Logical Instructions

Mne-monic	Operation	Boolean/Arith Operation	Flags NV-BDIZC	Address Mode	Assembler Notation	Op	~	#	Notes
AND	logical AND	A AND M → A	Nx-xxxZx	Immediate	AND #$dd	29	2	2	
				Absolute	AND $aaaa	2D	4	3	
				Zero page	AND $aa	25	3	2	
				Indxd Indct	AND ($ff,X)	21	6	2	
				Indct Indxd	AND ($aa),Y	31	5	2	
				Zero page,X	AND $ff,X	35	4	2	
				Absolute,X	AND $ffff,X	3D	4*	3	
				Absolute,Y	AND $ffff,Y	39	4*	3	

Mne-monic	Operation	Boolean/Arith Operation	Flags NV-BDIZC	Address Mode	Assembler Notation	Op	~	#	Notes
EOR	Exclusive **OR**	A EOR M → A	Nx-xxxZx	Immediate	EOR #$dd	49	2	2	
				Absolute	EOR $aaaa	4D	4	3	
				Zero page	EOR $aa	45	3	2	
				Indxd Indct	EOR ($ff,X)	41	6	2	
				Indct Indxd	EOR ($aa),Y	51	5*	2	
				Zero page,X	EOR $ff,X	55	4	2	
				Absolute,X	EOR $ffff,X	5D	4*	3	
				Absolute,Y	EOR $ffff,Y	59	4*	3	
ORA	**OR A**ccumulator	A OR M → A	Nx-xxxZx	Immediate	ORA #$dd	09	2	2	
				Absolute	ORA $aaaa	0D	4	3	
				Zero page	ORA $aa	05	3	2	
				Indxd Indct	ORA ($ff,X)	01	6	2	
				Indct Indxd	ORA ($aa),Y	11	5	2	
				Zero page,X	ORA $ff,X	15	4	2	
				Absolute,X	ORA $ffff,X	1D	4*	3	
				Absolute,Y	ORA $ffff,Y	19	4*	3	
BIT	test memory **BIT**s	A AND M $M_7 \rightarrow N$ $M_6 \rightarrow V$	76-xxxZx	Absolute	BIT $aaaa	2C	4	3	Memory bits 7 and 6 are transferred into the N and V flags respectively.
				Zero page	BIT $aa	24	3	2	

Rotate and Shift Instructions

Mne-monic	Operation	Boolean/Arith Operation	Flags NV-BDIZC	Address Mode	Assembler Notation	Op	~	#	Notes
ASL	Arithmetic Shift Left	C ← 7...0 ← 0	Nx-xxxZC	Absolute	ASL $aaaa	0E	6	3	
				Zero page	ASL $aa	06	5	2	
				Accumulator	ASL A	0A	2	1	
				Zero page,X	ASL $ff,X	16	6	2	
				Absolute,X	ASL $ffff,X	1E	7	3	
LSR	Logical Shift Right	0 → 7...0 → C	0x-xxxZC	Absolute	LSR $aaaa	4E	6	3	
				Zero page	LSR $aa	46	5	2	
				Accumulator	LSR A	4A	2	1	
				Zero page,X	LSR $ff,X	56	6	2	
				Absolute,X	LSR $ffff,X	5E	7	3	
ROL	**RO**tate **L**eft		Nx-xxxZC	Absolute	ROL $aaaa	2E	6	3	
				Zero page	ROL $aa	26	5	2	
				Accumulator	ROL A	2A	2	1	
				Zero page,X	ROL $ff,X	36	6	2	
				Absolute,X	ROL $ffff,X	3E	7	3	
ROR	**RO**tate **R**ight		Nx-xxxZC	Absolute	ROR $aaaa	6E	6	3	
				Zero page	ROR $aa	66	5	2	
				Accumulator	ROR A	6A	2	1	
				Zero page,X	ROR $ff,X	76	6	2	
				Absolute,X	ROR $ffff,X	7E	7	3	

Increment and Decrement Instructions

Mne-monic	Operation	Boolean/Arith Operation	Flags NV-BDIZC	Address Mode	Assembler Notation	Op	~	#	Notes
INC	**INC**rement memory	M + 1 → M	Nx-xxxZx	Absolute	INC $aaaa	EE	6	3	
				Zero page	INC $aa	E6	5	2	
				Zero page,X	INC $ff,X	F6	6	2	
				Absolute,X	INC $ffff,X	FE	7	3	

EXPANDED TABLE OF 6502 INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Mne-monic	Operation	Boolean/Arith Operation	Flags NV-BDIZC	Address Mode	Assembler Notation	Op	~	#	Notes
INX	INcrement X register	$X + 1 \rightarrow X$	Nx-xxxZx	Implied	INX	E8	2	1	
INY	INcrement Y register	$Y + 1 \rightarrow Y$	Nx-xxxZx	Implied	INY	C8	2	1	
DEC	DECrement memory	$M - 1 \rightarrow M$	Nx-xxxZx	Absolute	DEC $aaaa	CE	6	3	
				Zero page	DEC $aa	C6	5	2	
				Zero page,X	DEC $ff,X	D6	6	2	
				Absolute,X	DEC $ffff,X	DE	7	3	
DEX	DEcrement X register	$X - 1 \rightarrow X$	Nx-xxxZx	Implied	DEX	CA	2	1	
DEY	DEcrement Y register	$Y - 1 \rightarrow Y$	Nx-xxxZx	Implied	DEY	88	2	1	

Unconditional Jump Instructions

Mne-monic	Operation	Boolean/Arith Operation	Flags NV-BDIZC	Address Mode	Assembler Notation	Op	~	#	Notes

JMP	JuMP to new memory location	aaaa → PC {abs addressing}
		
		(aaaa) → PC_L
		(aaaa + 1) → PC_H
		{indirect addressing}
			xx-xxxxx	Absolute	JMP $aaaa	4C	3	3	In the indirect addressing mode, aaaa is not transferred into the PC but rather the contents of memory location aaaa and aaaa+1 are placed in the PC.
									Special Note: Care should be used with this mode because of a bug in the 6502 chip family. If the indirect address is located at a page boundary {example, JMP ($5FFF)} an incorrect address will be generated.

				Indirect	JMP ($aaaa)	6C	5	3	

Test (Compare) Instructions

Mne-monic	Operation	Boolean/Arith Operation	Flags NV-BDIZC	Address Mode	Assembler Notation	Op	~	#	Notes
CMP	CoMPare memory location to accumulator	A - M	Nx-xxxZC	Immediate	CMP #$dd	C9	2	2	
				Absolute	CMP $aaaa	CD	4	3	
				Zero page	CMP $aa	C5	3	2	
				Indxd Indct	CMP ($ff,X)	C1	6	2	
				Indct Indxd	CMP ($aa),Y	D1	5*	2	
				Zero page,X	CMP $ff,X	D5	4	2	
				Absolute,X	CMP $ffff,X	DD	4*	3	
				Absolute,Y	CMP $ffff,Y	D9	4*	3	
CPX	ComPare memory location to X register	X - M	Nx-xxxZC	Immediate	CPX #$dd	E0	2	2	
				Absolute	CPX $aaaa	EC	4	3	
				Zero page	CPX $aa	E4	3	2	
CPY	ComPare memory location to Y register	Y - M	Nx-xxxZC	Immediate	CPY #$dd	C0	2	2	
				Absolute	CPY $aaaa	CC	4	3	
				Zero page	CPY $aa	C4	3	2	

Mne-monic	Operation	Boolean/Arith Operation	Flags NV-BDIZC	Address Mode	Assembler Notation	Op	~	#	Notes

Conditional Jump (Branch) Instructions

Mne-monic	Operation	Boolean/Arith Operation	Flags NV-BDIZC	Address Mode	Assembler Notation	Op	~	#	Notes
BCC	Branch if Carry Clear	PC + rr → PC if C=0	xx-xxxxx	Relative	BCC $rr	90	2+2		
BCS	Branch if Carry Set	PC + rr → PC if C=1	xx-xxxxx	Relative	BCS $rr	B0	2+2		
BEQ	Branch if last result EQual to zero	PC + rr → PC if Z=1	xx-xxxxx	Relative	BEQ $rr	F0	2+2		
BNE	Branch if last result Not Equal to zero	PC + rr → PC if Z=0	xx-xxxxx	Relative	BNE $rr	D0	2+2		
BMI	Branch if last result a MInus (neg) number	PC + rr → PC if N=1	xx-xxxxx	Relative	BMI $rr	30	2+2		
BPL	Branch is last result a PLus (pos) number	PC + rr → PC if N=0	xx-xxxxx	Relative	BPL $rr	10	2+2		
BVC	Branch if oVerflow flag Clear	PC + rr → PC if V=0	xx-xxxxx	Relative	BVC $rr	50	2+2		
BVS	Branch if oVerflow flag Set	PC + rr → PC if V=1	xx-xxxxx	Relative	BVS $rr	70	2+2		

Subroutine Instructions

Mne-monic	Operation	Boolean/Arith Operation	Flags NV-BDIZC	Address Mode	Assembler Notation	Op	~	#	Notes
JSR	Jump to SubRoutine	PC + 2 → S aaaa → PC SP - 2 → SP	xx-xxxxx	Absolute	JSR $aaaa	20	6	3	
RTS	ReTurn from Subroutine	S (2 bytes) → PC PC + 1 → PC SP + 2 → SP	xx-xxxxx	Implied	RTS	60	6	1	

Stack Instructions

Mne-monic	Operation	Boolean/Arith Operation	Flags NV-BDIZC	Address Mode	Assembler Notation	Op	~	#	Notes
PHA	PusH Accumulator onto stack	A → S SP - 1 → SP	xx-xxxxx	Implied	PHA	48	3	1	
PLA	PulL Accumulator from stack	S → A SP + 1 → SP	Nx-xxxZx	Implied	PLA	68	4	1	
PHP	PusH Processor status register onto stack	PSR → S SP - 1 → SP	xx-xxxxx	Implied	PHP	08	3	1	

EXPANDED TABLE OF 6502 INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Mne-monic	Operation	Boolean/Arith Operation	Flags NV-BDIZC	Address Mode	Assembler Notation	Op	~	#	Notes
PLP	PulL Processor status register from stack	S → PSR SP + 1 → SP	NV-BDIZC	Implied	PLP	28	4	1	
TXS	Transfer X register into Stack pointer	X → SP	xx-xxxxx	Implied	TXS	9A	2	1	
TSX	Transfer Stack pointer into X register	SP → X	Nx-xxxZx	Implied	TSX	BA	2	1	
			Interrupt Instructions						
RTI	ReTurn from Interrupt	S → PSR SP + 1 → SP S (2 bytes) → PC SP + 2 → SP	NV-BDIZC	Implied	RTI	40	6	1	
			Input-Output Instructions						
none									The 6502 memory-maps all input and output rather than using special instructions.

Notes

Address Modes	Assembler Notation
Immediate	Mnemonic #$dd
Absolute	Mnemonic $aaaa
Zero page	Mnemonic $aa
Accumulator	Mnemonic A
Implied	Mnemonic
Indxd Indct	Mnemonic ($ff,X)
Indct Indxd	Mnemonic ($aa),Y
Zero page,X	Mnemonic $ff,X
Absolute,X	Mnemonic $ffff,X
Absolute,Y	Mnemonic $ffff,Y
Relative	Mnemonic $rr
Indirect	Mnemonic ($aaaa)
Zero page,Y	Mnemonic $ff,Y

Abbreviations and Explanations

Indxd Indct = Indexed Indirect
Indct Indxd = Indirect Indexed
a = address (one hex digit)
d = data (one hex digit)
f = address offset (one hex digit) ($ff is an unsigned binary number and is therefore positive)
r = relative address (one hex digit) ($rr is a 2's-complement signed binary number and can therefore be positive or negative)
* = add 1 cycle if page boundary crossed
+ = add 1 cycle if branch occurs; add 1 more cycle if branch crosses page
( ) =the contents of the address within parentheses form the actual address
7...0 = bits 0 through 7 of memory or the accumulator
M_7, M_6, etc. = Bits 7, 6, etc. of a memory location
$_L$ = low-order byte
$_H$ = high-order byte

PC = program counter
S = stack (contents of the top byte of the stack)
SP = stack pointer
PSR = processor status register (flags)
* = Add 1 cycle if crossing page boundary

Flags

0 = flag always cleared
1 = flag always set

x = flag not affected
N = negative flag
V = overflow flag
B = break flag
D = decimal flag
I = interrupt flag
Z = zero flag
C = carry flag

Symbols in the Page Heading

~ = clock cycles
= # of bytes used by instruction (and following address or data if used)

Addressing Modes - Summary

Immediate (Mnemonic #$dd): The data to be operated on (#$dd) is in the next byte of memory after the instruction itself. Therefore no address is needed.

Absolute (Mnemonic $aaaa): The data to be operated on is found in the memory location indicated ($aaaa). This is a 2-byte address and can point to any place in the 6502's 64K (65,536 byte) addressing range.

Zero page (Mnemonic $aa): The data to be operated on is found in the memory location indicated ($aa). This is a 1-byte address and can point only to a place in page zero of memory. Page zero is address $00-$FF (decimal 0-255).

Accumulator (Mnemonic A): These are instructions which use implied addressing, where the data is already in the accumulator.

Implied (Mnemonic): These instructions indicate where the data is or will be within the instruction itself.

Indxd Indct (Mnemonic ($ff,X)): In this form of addressing, the operand (the number which is going to have something done to it) is found through a multistep process. First, the offset ($ff) is added to the X register to form an address (this address must be in page zero since both of these are 8-bit numbers). The microprocessor then gets the contents of this memory location and the following location to form another address where it will then find the data (operand).

Indct Indxd (Mnemonic ($aa),Y): This addressing mode is sometimes confused with the one above though it does work differently. First, the microprocessor goes to address $aa and the address immediately following $aa. It uses the contents of these two locations to form a 16-bit address to which the Y register is added. This then forms the actual address where the operand is located.

Zero page,X (Mnemonic $ff,X): In this form of addressing the number $ff is added to the X register to form a second address where the operand is located. Because both $ff and X are 8-bit binary numbers, the actual address must be in page zero. If the sum of these two numbers exceeds $FF (the end of page zero), any carry will be ignored and the address will "wrap around" to the beginning of page zero.

Absolute,X (Mnemonic $ffff,X): In this case, the 16-bit number $ffff is added to the X register to form the actual address. If this number exceeds hexadecimal $FFFF, the carry is ignored and the address "wraps around" to $0000 and continues from there.

Absolute,Y (Mnemonic $ffff,Y): This address mode works the same as Absolute,X except that the Y register is used instead.

Relative (Mnemonic $rr): $rr is a 2's-complement signed binary number; that is, it can be positive or negative. This number is added to the current contents of the program counter to determine the actual address. $rr is different from an offset ($ffff or $ff) because it is not added to another register but directly to the program counter itself. It directs the microprocessor relative to its current place in memory.

Indirect (Mnemonic ($aaaa)): In this mode, the contents of address $aaaa and the contents of the address immediately following it are used to form the actual address where the operand is to be found. (Only the JMP instruction uses this addressing mode.)

Zero page,Y (Mnemonic $ff,Y): This addressing mode is exactly like the "Zero page,X" mode except that register Y is used instead.

SHORT TABLE OF 6502 INSTRUCTIONS LISTED BY CATEGORY

CPU Control Instructions

Assembler Notation	Op	Boolean/Arith Operation	Flags NV-BDIZC
NOP	EA	Nothing	xx-xxxxx
BRK	00	PC + 2 → S SP - 2 → SP PSR → S SP - 1 → S $FFFE → PC	xx-1x1xx

Data Transfer Instructions

Assembler Notation	Op	Boolean/Arith Operation	Flags NV-BDIZC
LDA #$dd	A9	M → A	Nx-xxxZx
LDA $aaaa	AD		
LDA $aa	A5		
LDA ($ff,X)	A1		
LDA ($aa),Y	B1		
LDA $ff,X	B5		
LDA $ffff,X	BD		
LDA $ffff,Y	B9		
LDX #$dd	A2	M → X	Nx-xxxZx
LDX $aaaa	AE		
LDX $aa	A6		
LDX $ffff,Y	BE		
LDX $ff,Y	B6		
LDY #$dd	A0	M → Y	Nx-xxxZx
LDY $aaaa	AC		
LDY $aa	A4		
LDY $ff,X	B4		
LDY $ffff,X	BC		
STA $aaaa	8D	A → M	xx-xxxxx
STA $aa	85		
STA ($ff,X)	81		
STA ($aa),Y	91		
STA $ff,X	95		
STA $ffff,X	9D		
STA $ffff,Y	99		
STX $aaaa	8E	X → M	xx-xxxxx
STX $aa	86		
STX $ff,Y	96		
STY $aaaa	8C	Y → M	xx-xxxxx
STY $aa	84		
STY $ff,X	94		
TAX	AA	A → X	Nx-xxxZx
TXA	8A	X → A	Nx-xxxZx
TAY	A8	A → Y	Nx-xxxZx
TYA	98	Y → A	Nx-xxxZx

Flag Instructions

Assembler Notation	Op	Boolean/Arith Operation	Flags NV-BDIZC
CLC	18	0 → C	xx-xxxx0
CLD	D8	0 → D	xx-x0xxx
CLI	58	0 → I	xx-xx0xx
CLV	B8	0 → V	x0-xxxxx
SEC	38	1 → C	xx-xxxx1
SED	F8	1 → D	xx-x1xxx
SEI	78	1 → I	xx-xx1xx

Arithmetic Instructions

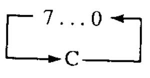
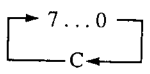
Assembler Notation	Op	Boolean/Arith Operation	Flags NV-BDIZC
ADC #$dd	69	A + M + C → A	NV-xxxZC
ADC $aaaa	6D		
ADC $aa	65		
ADC ($ff,X)	61		
ADC ($aa),Y	71		
ADC $ff,X	75		
ADC $ffff,X	7D		
ADC $ffff,Y	79		
SBC #$dd	E9	A - M -	NV-xxxZC
SBC $aaaa	ED	(1-C) → A	
SBC $aa	E5		
SBC ($ff,X)	E1	Note: (1-C) =	
SBC ($aa),Y	F1	Borrow	
SBC $ff,X	F5		
SBC $ffff,X	FD		
SBC $ffff,Y	F9		

Logical Instructions

Assembler Notation	Op	Boolean/Arith Operation	Flags NV-BDIZC
AND #$dd	29	A AND M → A	Nx-xxxZx
AND $aaaa	2D		
AND $aa	25		
AND ($ff,X)	21		
AND ($aa),Y	31		
AND $ff,X	35		
AND $ffff,X	3D		
AND $ffff,Y	39		

Assembler Notation	Op	Boolean/Arith Operation	Flags NV-BDIZC
EOR #$dd	49	A EOR M → A	Nx-xxxZx
EOR $aaaa	4D		
EOR $aa	45		
EOR ($ff,X)	41		
EOR ($aa),Y	51		
EOR $ff,X	55		
EOR $ffff,X	5D		
EOR $ffff,Y	59		
ORA #$dd	09	A OR M → A	Nx-xxxZx
ORA $aaaa	0D		
ORA $aa	05		
ORA ($ff,X)	01		
ORA ($aa),Y	11		
ORA $ff,X	15		
ORA $ffff,X	1D		
ORA $ffff,Y	19		
BIT $aaaa	2C	A AND M $M_7 \rightarrow N$ $M_6 \rightarrow V$	76-xxxZx
BIT $aa	24		

Rotate and Shift Instructions

ASL $aaaa	0E	C ← 7...0 ← 0	Nx-xxxZC
ASL $aa	06		
ASL A	0A		
ASL $ff,X	16		
ASL $ffff,X	1E		
LSR $aaaa	4E	0 → 7...0 → C	0x-xxxZC
LSR $aa	46		
LSR A	4A		
LSR $ff,X	56		
LSR $ffff,X	5E		
ROL $aaaa	2E		Nx-xxxZC
ROL $aa	26		
ROL A	2A		
ROL $ff,X	36		
ROL $ffff,X	3E		
ROR $aaaa	6E		Nx-xxxZC
ROR $aa	66		
ROR A	6A		
ROR $ff,X	76		
ROR $ffff,X	7E		

Increment and Decrement Instructions

Assembler Notation	Op	Boolean/Arith Operation	Flags NV-BDIZC
INC $aaaa	EE	M + 1 → M	Nx-xxxZx
INC $aa	E6		
INC $ff,X	F6		
INC $ffff,X	FE		
INX	E8	X + 1 → X	Nx-xxxZx
INY	C8	Y + 1 → Y	Nx-xxxZx
DEC $aaaa	CE	M - 1 → M	Nx-xxxZx
DEC $aa	C6		
DEC $ff,X	D6		
DEC $ffff,X	DE		
DEX	CA	X - 1 → X	Nx-xxxZx
DEY	88	Y - 1 → Y	Nx-xxxZx

Unconditional Jump Instructions

JMP $aaaa	4C	aaaa → PC {abs addressing}	xx-xxxxx
JMP ($aaaa)	6C	(aaaa) → PC_L (aaaa + 1) → PC_H {indirect addressing}	

Test (Compare) Instructions

CMP #$dd	C9	A - M	Nx-xxxZC
CMP $aaaa	CD		
CMP $aa	C5		
CMP ($ff,X)	C1		
CMP ($aa),Y	D1		
CMP $ff,X	D5		
CMP $ffff,X	DD		
CMP $ffff,Y	D9		
CPX #$dd	E0	X - M	Nx-xxxZC
CPX $aaaa	EC		
CPX $aa	E4		
CPY #$dd	C0	Y - M	Nx-xxxZC
CPY $aaaa	CC		
CPY $aa	C4		

Conditional Jump (Branch) Instructions

BCC $rr	90	PC + rr → PC if C=0	xx-xxxxx
BCS $rr	B0	PC + rr → PC if C=1	xx-xxxxx
BEQ $rr	F0	PC + rr → PC if Z=1	xx-xxxxx

SHORT TABLE OF 6502 INSTRUCTIONS LISTED BY CATEGORY *(Continued)*

Assembler Notation	Op	Boolean/Arith Operation	Flags NV-BDIZC
BNE $rr	D0	PC + rr → PC if Z=0	xx-xxxxx
BMI $rr	30	PC + rr → PC if N=1	xx-xxxxx
BPL $rr	10	PC + rr → PC if N=0	xx-xxxxx
BVC $rr	50	PC + rr → PC if V=0	xx-xxxxx
BVS $rr	70	PC + rr → PC if V=1	xx-xxxxx
Subroutine Instructions			
JSR $aaaa	20	PC + 2 → S aaaa → PC SP - 2 → SP	xx-xxxxx
RTS	60	S (2 bytes) → PC PC + 1 → PC SP + 2 → SP	xx-xxxxx
Stack Instructions			
PHA	48	A → S SP - 1 → SP	xx-xxxxx
PLA	68	S → A SP + 1 → SP	Nx-xxxZx
PHP	08	PSR → S SP - 1 → SP	xx-xxxxx
PLP	28	S → PSR SP + 1 → SP	NV-BDIZC
TXS	9A	X → SP	xx-xxxxx
TSX	BA	SP → X	Nx-xxxZx
Interrupt Instructions			
RTI	40	S → PSR SP + 1 → SP S (2 bytes) → PC SP + 2 → SP	NV-BDIZC
Input-Output Instructions			
None			

CONDENSED TABLE OF 6502 INSTRUCTIONS LISTED BY CATEGORY

CPU Control Instructions	
NOP	EA
BRK	00
Data Transfer Instructions	
LDA #$dd	A9
LDA $aaaa	AD
LDA $aa	A5
LDA ($ff,X)	A1
LDA ($aa),Y	B1
LDA $ff,X	B5
LDA $ffff,X	BD
LDA $ffff,Y	B9
LDX #$dd	A2
LDX $aaaa	AE
LDX $aa	A6
LDX $ffff,Y	BE
LDX $ff,Y	B6
LDY #$dd	A0
LDY $aaaa	AC
LDY $aa	A4
LDY $ff,X	B4
LDY $ffff,X	BC
STA $aaaa	8D
STA $aa	85
STA ($ff,X)	81
STA ($aa),Y	91
STA $ff,X	95
STA $ffff,X	9D
STA $ffff,Y	99
STX $aaaa	8E
STX $aa	86
STX $ff,Y	96
STY $aaaa	8C
STY $aa	84
STY $ff,X	94
TAX	AA
TXA	8A
TAY	A8
TYA	98
Flag Instructions	
CLC	18
CLD	D8
CLI	58
CLV	B8
SEC	38
SED	F8
SEI	78
Arithmetic Instructions	
ADC #$dd	69
ADC $aaaa	6D
ADC $aa	65
ADC ($ff,X)	61

CONDENSED TABLE OF 6502 INSTRUCTIONS LISTED BY CATEGORY (*Continued*)

ADC ($aa),Y	71
ADC $ff,X	75
ADC $ffff,X	7D
ADC $ffff,Y	79
SBC #$dd	E9
SBC $aaaa	ED
SBC $aa	E5
SBC ($ff,X)	E1
SBC ($aa),Y	F1
SBC $ff,X	F5
SBC $ffff,X	FD
SBC $ffff,Y	F9
Logical Instructions	
AND #$dd	29
AND $aaaa	2D
AND $aa	25
AND ($ff,X)	21
AND ($aa),Y	31
AND $ff,X	35
AND $ffff,X	3D
AND $ffff,Y	39
EOR #$dd	49
EOR $aaaa	4D
EOR $aa	45
EOR ($ff,X)	41
EOR ($aa),Y	51
EOR $ff,X	55
EOR $ffff,X	5D
EOR $ffff,Y	59
ORA #$dd	09
ORA $aaaa	0D
ORA $aa	05
ORA ($ff,X)	01
ORA ($aa),Y	11
ORA $ff,X	15
ORA $ffff,X	1D
ORA $ffff,Y	19
BIT $aaaa	2C
BIT $aa	24
Rotate and Shift Instructions	
ASL $aaaa	0E
ASL $aa	06
ASL A	0A
ASL $ff,X	16
ASL $ffff,X	1E
LSR $aaaa	4E
LSR $aa	46
LSR A	4A
LSR $ff,X	56
LSR $ffff,X	5E
ROL $aaaa	2E
ROL $aa	26
ROL A	2A
ROL $ff,X	36
ROL $ffff,X	3E
ROR $aaaa	6E
ROR $aa	66
ROR A	6A
ROR $ff,X	76
ROR $ffff,X	7E
Increment and Decrement Instructions	
INC $aaaa	EE
INC $aa	E6
INC $ff,X	F6
INC $ffff,X	FE
INX	E8
INY	C8
DEC $aaaa	CE
DEC $aa	C6
DEC $ff,X	D6
DEC $ffff,X	DE
DEX	CA
DEY	88
Unconditional Jump Instructions	
JMP $aaaa	4C
JMP ($aaaa)	6C
Test (Compare) Instructions	
CMP #$dd	C9
CMP $aaaa	CD
CMP $aa	C5
CMP ($ff,X)	C1
CMP ($aa),Y	D1
CMP $ff,X	D5
CMP $ffff,X	DD
CMP $ffff,Y	D9
CPX #$dd	E0
CPX $aaaa	EC
CPX $aa	E4
CPY #$dd	C0
CPY $aaaa	CC
CPY $aa	C4
Conditional Jump (Branch) Instructions	
BCC $rr	90
BCS $rr	B0
BEQ $rr	F0
BNE $rr	D0
BMI $rr	30
BPL $rr	10
BVC $rr	50
BVS $rr	70
Subroutine Instructions	
JSR $aaaa	20
RTS	60
Stack Instructions	
PHA	48
PLA	68
PHP	08
PLP	28
TXS	9A
TSX	BA
Interrupt Instructions	
RTI	40
Input-Output Instructions	
None	

CONDENSED TABLE OF 6502 INSTRUCTIONS LISTED ALPHABETICALLY

ADC ($aa),Y	71
ADC ($ff,X)	61
ADC $aa	65
ADC $aaaa	6D
ADC $ffff,X	7D
ADC $ffff,Y	79
ADC $ff,X	75
ADC #$dd	69
AND ($aa),Y	31
AND ($ff,X)	21
AND $aa	25
AND $aaaa	2D
AND $ffff,X	3D
AND $ffff,Y	39
AND $ff,X	35
AND #$dd	29
ASL $aa	06
ASL $aaaa	0E
ASL $ffff,X	1E
ASL $ff,X	16
ASL A	0A
BCC $rr	90
BCS $rr	B0
BEQ $rr	F0
BIT $aa	24
BIT $aaaa	2C
BMI $rr	30
BNE $rr	D0
BPL $rr	10
BRK	00
BVC $rr	50
BVS $rr	70
CLC	18
CLD	D8
CLI	58
CLV	B8
CMP ($aa),Y	D1
CMP ($ff,X)	C1
CMP $aa	C5
CMP $aaaa	CD
CMP $ffff,X	DD
CMP $ffff,Y	D9
CMP $ff,X	D5
CMP #$dd	C9
CPX $aa	E4
CPX $aaaa	EC
CPX #$dd	E0
CPY $aa	C4
CPY $aaaa	CC
CPY #$dd	C0
DEC $aa	C6
DEC $aaaa	CE

CONDENSED TABLE OF 6502 INSTRUCTIONS LISTED ALPHABETICALLY (*Continued*)

DEC $ffff,X	DE	LDA $ffff,X	BD	ORA $ffff,Y	19	SBC $ff,X	F5
DEC $ff,X	D6	LDA $ffff,Y	B9	ORA $ff,X	15	SBC #$dd	E9
DEX	CA	LDA $ff,X	B5	ORA #$dd	09	SEC	38
DEY	88	LDA #$dd	A9	PHA	48	SED	F8
EOR ($aa),Y	51	LDX $aa	A6	PHP	08	SEI	78
EOR ($ff,X)	41	LDX $aaaa	AE	PLA	68	STA ($aa),Y	91
EOR $aa	45	LDX $ffff,Y	BE	PLP	28	STA ($ff,X)	81
EOR $aaaa	4D	LDX $ff,Y	B6	ROL $aa	26	STA $aa	85
EOR $ffff,X	5D	LDX #$dd	A2	ROL $aaaa	2E	STA $aaaa	8D
EOR $ffff,Y	59	LDY $aa	A4	ROL $ffff,X	3E	STA $ffff,X	9D
EOR $ff,X	55	LDY $aaaa	AC	ROL $ff,X	36	STA $ffff,Y	99
EOR #$dd	49	LDY $ffff,X	BC	ROL A	2A	STA $ff,X	95
INC $aa	E6	LDY $ff,X	B4	ROR $aa	66	STX $aa	86
INC $aaaa	EE	LDY #$dd	A0	ROR $aaaa	6E	STX $aaaa	8E
INC $ffff,X	FE	LSR $aa	46	ROR $ffff,X	7E	STX $ff,Y	96
INC $ff,X	F6	LSR $aaaa	4E	ROR $ff,X	76	STY $aa	84
INX	E8	LSR $ffff,X	5E	ROR A	6A	STY $aaaa	8C
INY	C8	LSR $ff,X	56	RTI	40	STY $ff,X	94
JMP ($aaaa)	6C	LSR A	4A	RTS	60	TAX	AA
JMP $aaaa	4C	NOP	EA	SBC ($aa),Y	F1	TAY	A8
JSR $aaaa	20	ORA ($aa),Y	11	SBC ($ff,X)	E1	TSX	BA
LDA ($aa),Y	B1	ORA ($ff,X)	01	SBC $aa	E5	TXA	8A
LDA ($ff,X)	A1	ORA $aa	05	SBC $aaaa	ED	TXS	9A
LDA $aa	A5	ORA $aaaa	0D	SBC $ffff,X	FD	TYA	98
LDA $aaaa	AD	ORA $ffff,X	1D	SBC $ffff,Y	F9		

CONDENSED TABLE OF 6502 INSTRUCTIONS LISTED BY OP CODE

00	BRK	31	AND ($aa),Y	68	PLA	99	STA $ffff,Y
01	ORA ($ff,X)	35	AND $ff,X	69	ADC #$dd	9A	TXS
05	ORA $aa	36	ROL $ff,X	6A	ROR A	9D	STA $ffff,X
06	ASL $aa	38	SEC	6C	JMP ($aaaa)	A0	LDY #$dd
08	PHP	39	AND $ffff,Y	6D	ADC $aaaa	A1	LDA ($ff,X)
09	ORA #$dd	3D	AND $ffff,X	6E	ROR $aaaa	A2	LDX #$dd
0A	ASL A	3E	ROL $ffff,X	70	BVS $rr	A4	LDY $aa
0D	ORA $aaaa	40	RTI	71	ADC ($aa),Y	A5	LDA $aa
0E	ASL $aaaa	41	EOR ($ff,X)	75	ADC $ff,X	A6	LDX $aa
10	BPL $rr	45	EOR $aa	76	ROR $ff,X	A8	TAY
11	ORA ($aa),Y	46	LSR $aa	78	SEI	A9	LDA #$dd
15	ORA $ff,X	48	PHA	79	ADC $ffff,Y	AA	TAX
16	ASL $ff,X	49	EOR #$dd	7D	ADC $ffff,X	AC	LDY $aaaa
18	CLC	4A	LSR A	7E	ROR $ffff,X	AD	LDA $aaaa
19	ORA $ffff,Y	4C	JMP $aaaa	81	STA ($ff,X)	AE	LDX $aaaa
1D	ORA $ffff,X	4D	EOR $aaaa	84	STY $aa	B0	BCS $rr
1E	ASL $ffff,X	4E	LSR $aaaa	85	STA $aa	B1	LDA ($aa),Y
20	JSR $aaaa	50	BVC $rr	86	STX $aa	B4	LDY $ff,X
21	AND ($ff,X)	51	EOR ($aa),Y	88	DEY	B5	LDA $ff,X
24	BIT $aa	55	EOR $ff,X	8A	TXA	B6	LDX $ff,Y
25	AND $aa	56	LSR $ff,X	8C	STY $aaaa	B8	CLV
26	ROL $aa	58	CLI	8D	STA $aaaa	B9	LDA $ffff,Y
28	PLP	59	EOR $ffff,Y	8E	STX $aaaa	BA	TSX
29	AND #$dd	5D	EOR $ffff,X	90	BCC $rr	BC	LDY $ffff,X
2A	ROL A	5E	LSR $ffff,X	91	STA ($aa),Y	BD	LDA $ffff,X
2C	BIT $aaaa	60	RTS	94	STY $ff,X	BE	LDX $ffff,Y
2D	AND $aaaa	61	ADC ($ff,X)	95	STA $ff,X	C0	CPY #$dd
2E	ROL $aaaa	65	ADC $aa	96	STX $ff,Y	C1	CMP ($ff,X)
30	BMI $rr	66	ROR $aa	98	TYA	C4	CPY $aa

CONDENSED TABLE OF 6502 INSTRUCTIONS LISTED BY OP CODE *(Continued)*

C5	CMP $aa	D1	CMP ($aa),Y	E4	CPX $aa	F0	BEQ $rr
C6	DEC $aa	D5	CMP $ff,X	E5	SBC $aa	F1	SBC ($aa),Y
C8	INY	D6	DEC $ff,X	E6	INC $aa	F5	SBC $ff,X
C9	CMP #$dd	D8	CLD	E8	INX	F6	INC $ff,X
CA	DEX	D9	CMP $ffff,Y	E9	SBC #$dd	F8	SED
CC	CPY $aaaa	DD	CMP $ffff,X	EA	NOP	F9	SBC $ffff,Y
CD	CMP $aaaa	DE	DEC $ffff,X	EC	CPX $aaaa	FD	SBC $ffff,X
CE	DEC $aaaa	E0	CPX #$dd	ED	SBC $aaaa	FE	INC $ffff,X
D0	BNE $rr	E1	SBC ($ff,X)	EE	INC $aaaa		