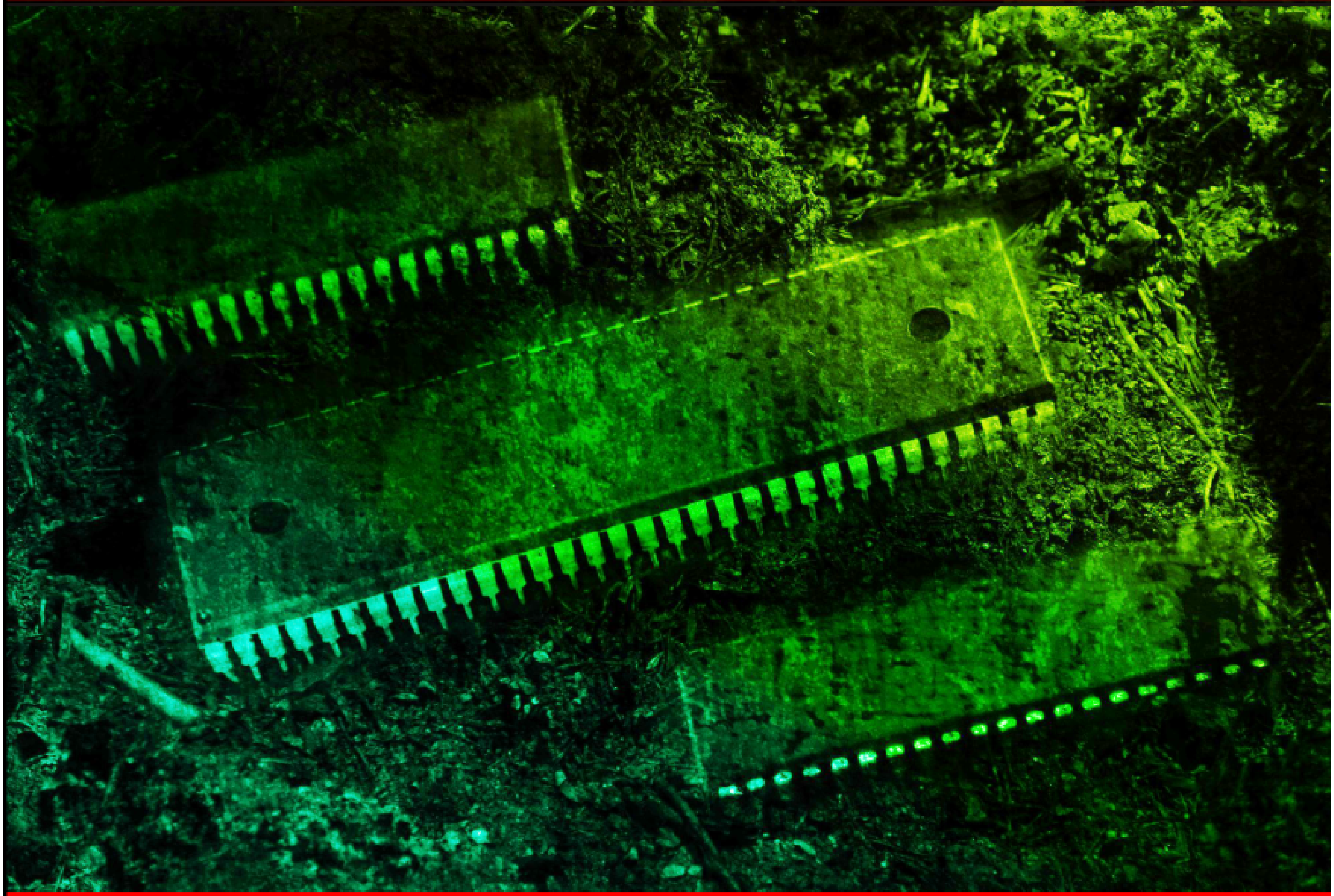
Learn Multiplatform Assembly Programming

with ChibiAkumas!



Cheatsheet Collection:

6502, 6280, 6309, 6809, 65816, 68000, 8086, ARM, ARM THUMB, IBM 370, MIPS, PDP-11, PowerPC, RISC-V, Super-H, TMS-9900, Z80, GBZ80, eZ80

www.LearnAsm.net

Z80

Mnemonic	Description	Example	Parameters	Flags affected
ADC r	Add register r and the carry flag to the Accumulator A.	ADC B	'r': (HL) (IX+#) (IY+#) A B C D E H L IXH IXL IYH IYL	S Z H V N C
ADC A,#	Add 8 bit number # and the carry to A.	ADC 128	'#': 0-255 ($00-$FF)	S Z H V N C
ADC HL,rr	Add 16 bit register rr and the carry to HL.	ADC HL,BC	'rr': BC DE HL SP	S Z H V N C
ADD rr2,rr1	Add 16 bit register rr1 to 16 bit register rr2.	ADD HL,BC	'rr1': HL IX IY 'rr2': BC DE SP *HL IX IY*	– – H – N C
ADD r	Adds 8 bit register r to A.	ADD B	'r': (HL) (IX+#) (IY+#) A B C D E H L IXH IXL IYH IYL	S Z H V N C
ADD #	Adds 8 bit value # to A.	ADD B	'#': 0-255 ($00-$FF)	S Z H V N C
AND r	Logical AND of bits in register r with Accumulator A.	AND B	'r': (HL) (IX+#) (IY+#) A B C D E H L IXH IXL IYH IYL	S Z H V N C
AND #	Logical AND of bits in 8 bit value # with Accumulator A.	AND $64	'#': 0-255 ($00-$FF)	S Z H V N C
BIT b,r	Test bit b from 8 bit register r and set the Z flag to that bit.	BIT 7,B	'b': 0-7 (%76543210) 'r': (HL) (IX+#) (IY+#) A B C D E H L	s Z H v N –
CALL addr	Call Subroutine at address addr	CALL $1000	'addr': 0-65535 ($0000-$FFFF)	– – – – – –
CALL c,addr	Call Subroutine at address addr only IF condition c is true.	CALL Z,$1000	'addr': 0-65535 ($0000-$FFFF) 'c': c m nc nz p po pe z	– – – – – –
CCF	Complement the Carry Flag. C flag will inverted	CCF		– – H – N C
CP r	Compare the Accumulator to register r.	CP B	'r': (HL) (IX+#) (IY+#) A B C D E H L	S Z H V N C
CP #	Compare the Accumulator to 8 bit immediate value #.	CP 32	'#':0-255 ($00-$FF)	S Z H V N C
CPD	Compare A to the byte at address HL and decrease HL and BC.	CPD		S Z H V N –
CPDR	Compare A to the byte at address HL and Decrease and Repeat	CPDR		S Z H V N –
CPI	Compare A to the byte at address HL and increase HL but decrease BC (Bytecount).	CPI		S Z H V N –
CPIR	Compare A to the byte at addr HL and inc HL dec BC (Bytecount) and Rep until match or BC=0.	CPIR		S Z H V N –
CPL	Invert all bits of A (this is known as 'One's Complement').	CPL		– – H – N –
DAA	Decimal Adjust Accumulator (Binary Coded Decimal)	DAA		S Z H V – C
DEC r	Decrease value in 8 bit register r by one.	DEC B	'r': (HL) (IX+#) (IY+#) A B C D E H L IXH IXL IYH IYL	S Z H V N –
DEC rr	Decrease value in 16 bit register rr by one.	DEC HL	Valid registers for 'rr': BC DE HL IX IY SP	– – – – – –
DI	Disable Maskable Interrupts	DI		– – – – – –
DJNZ ofst	Decrease B and Jump if NonZero to address offset #.	DJNZ label	'ofst': -128 to +127	– – – – – –
EI	Enable Maskable Interrupts.	EI		– – – – – –
EX (SP),HL	Exchange HL with the top item of the stack	EX (SP),HL		– – – – – –
EX AF,AF'	Exchange the Accumulator and Flags with the shadow Accumulator and Flags.	EX AF,AF'		S Z H V N C
EX DE,HL	Exchange HL and DE	EX DE,HL		– – – – – –
EXX	Exchange the registers BC, DE and HL with the shadow registers	EXX		– – – – – –
HALT	Stop the CPU until an interrupt occurs.	HALT		– – – – – –
IM0	Enable Interrupt mode 0.	IM0		– – – – – –
IM1	Enable Interrupt mode 1.	IM1		– – – – – –
IM2	Enable Interrupt mode 2.	IM2		– – – – – –
IN A,(#)	Read in an 8 bit byte A from 8 bit port #.	IN A,($10)	'#': 0-255 ($00-$FF)	S Z H V N –
IN r,(C)	Read in an 8 bit byte into register r from port (C)	IN A,(C)	'r': A B C D E H L	S Z H V N –
INC r	Increase value in 8 bit register r by one.	INC B	'r': (HL) (IX+#) (IY+#) A B C D E H L IXH IXL IYH IYL	S Z H V N –
INC rr	Increase value in 16 bit register rr by one.	INC HL	'rr': BC DE HL IX IY SP	– – – – – –
IND	Read a byte IN from port (C) and save to address in HL, then Decrease HL and B.	IND		s Z h v N –
INDR	Read a byte IN from port (C) and save to address in HL. then Decrease HL and B, rep until B=0.	INDR		s Z h v N –
INI	Read a byte IN from port (C) and save to address in HL, then increase HL and decrease B.	INI		s Z h v N –
INIR	Read a byte IN from port (C) and save to the address in HL, inc HL and dec B, rep until B=0.	INIR		s Z h v N –
JP (HL)	Jump to the address in register HL.	JP (HL)		– – – – – –
JP addr	Jump to the 16 bit address addr.	JP $4000	'addr': 0-65535 ($0000-$FFFF)	– – – – – –
JP c,addr	Jump to the 16 bit address addr only IF condition c is true in the flags register.	JP Z,$4000	'addr': 0-65535 ($0000-$FFFF) 'c': c m nc nz p po pe z	– – – – – –
JR ofst	Jump to the 8 bit offset #.	JR TestLabel	'#': -128 to +127	– – – – – –
JR c,ofst	Jump to the 8 bit offset ofst IF condition c is true.	JR Z,TestLabel	'ofst': -128 to +127	– – – – – –
LD (rr),A	Load the 8 bit value in the Accumulator into the address in register rr.	LD (DE),A	'rr': BC DE HL IX+# IY SP	– – – – – –
LD (HL),B	Load the 8 bit value in register r into the address in register rr.	LD (HL),B	'r': A B C D E H L 'rr': HL IX+# IY+#	– – – – – –
LD (addr),A	Load the 8 bit value in the Accumulator into memory address addr.	LD ($C000),A	'addr': 0-65535 ($0000-$FFFF)	– – – – – –
LD (addr),rr	Load the 16 bit value in register pair rr into memory address addr.	LD ($C000),BC	'addr': 0-65535 ($0000-$FFFF) 'rr': BC DE HL IX IY SP	– – – – – –
LD A,(rr)	Load the 8 bit value from the address in register rr into the Accumulator.	LD A,(DE)	'rr': BC DE HL IX+# IY SP	– – – – – –
LD A,(addr)	Load the 8 bit value from memory address addr into the Accumulator.	LD A,($C000)	'##': 0-65535 ($0000-$FFFF)	– – – – – –
LD r,#	Load the 8 bit register r with value #.	LD B,32	'r': A B C D E H L IXH IXL IYH IYL '#': 0-255 ($00-$FF)	– – – – – –
LD A,I	Load the 8 bit value from the I register to the Accumulator.	LD A,I		S Z H V N –
LD A,R	Load the 8 bit value from the R register to the Accumulator.	LD A,R		S Z H V N –
LD rr,(addr)	Load the 16 bit register pair rr from memory address addr.	LD BC,($C000)	'rr': BC DE HL IX IY SP 'addr': 0-65535 ($0000-$FFFF)	– – – – – –
LD rr,####	Load the 16 bit register pair rr with immediate value ####	LD BC,$C000	'rr': BC DE HL IX IY SP 'addr': 0-65535 ($0000-$FFFF)	– – – – – –
LD I,A	Load the 8 bit value from the Accumulator into the I register.	LD I,A		– – – – – –
LD R,A	Load the R register with the 8 bit value in the Accumulator.	LD R,A		– – – – – –
LD SP,HL	Load the 16 bit Stack Pointer register SP with the value in HL.	LD SP,HL		– – – – – –
LD r1,r2	Load the 8 bit register r1 from register r2.	LD H,B	'r1' and 'r2': A B C D E H L IXH IXL IYH IYL	– – – – – –
LD r,(rr)	Load the 8 bit register r from the address in register rr.	LD B,(HL)	'r': A B C D E H L 'rr': HL IX+# IY+#	– – – – – –
LDD	Load and Decrement. Copies bytes down from HL to DE with BC as a byte count.	LDD		– – H V N –
LDDR	Load, Decrement and Repeat. Copies bytes down from HL to DE with BC as a Byte count	LDDR		– – H V N –
LDI	Load and Increment. Copies bytes upwards from HL to DE with BC as a byte count	LDI		– – H V N –
LDIR	Load, Decrement and Repeat. Copies bytes upwards from HL to DE with BC as byte count	LDIR		– – H V N –
NEG	Negate the 8 bit value in the accumulator (Two's Complement of the number).	NEG		S Z H V N C
NOP	No Operation. This command has no effect on any registers or memory.	NOP		– – – – – –
OR r	Logical OR of bits in register r with Accumulator A.	OR B	'r': (HL) (IX+#) (IY+#) A B C D E H L IXH IXL IYH IYL	S Z H V N C
OR #	Logical OR of bits in 8 bit value # with Accumulator A.	OR $64	'#': 0-255 ($00-$FF)	S Z H V N C
OTDR	Out Decrement Repeat. Transfers B bytes from HL to port (C) moving downwards.	OTDR		s Z h v N –
OTIR	Out Increment Repeat. This command transfers B bytes from HL to port (C) moving upwards.	OTIR		s Z h v N –
OUT (#),A	Output an 8 bit byte from A to 8 bit port #.	OUT ($10),A	'#': 0-255 ($00-$FF)	– – – – – –
OUT (C),r	On a system with 8 bit ports, this will output an 8 bit byte from register r to port (C) .	OUT (C),r	'r': A B C D E H L	– – – – – –
OUT (C),0	On a system with 8 bit ports, this will output an 8 bit byte zero to port (C).	OUT (C),0		– – – – – –
OUTD	Out and Decrement. This command transfers a byte from HL to port (C) moving downwards.	OUTD		s Z h v N –
OUTI	Out and Increment. This command transfers a byte from HL to port (C) moving upwards.	OUTI		s Z h v N –
POP rr	Pop a pair of bytes off the stack into 16 bit register rr.	POP AF	'rr': AF BC DE HL IX IY	all if AF / none
PUSH rr	Push a pair of bytes from 16 bit register rr onto the top of the stack.	PUSH AF	'rr': AF BC DE HL IX IY	– – – – – –
RES b,r	Reset bit b from 8 bit register r to 0.	RES 7,B	'b': 0-7 (%76543210) 'r': (HL) (IX+#) (IY+#) A B C D E H L	– – – – – –
RET	Return from a subroutine.	RET		– – – – – –
RET c	Return from a subroutine only if condition c is true.	RET Z	'c': c m nc nz p po pe z	– – – – – –
RETI	Return from an interrupt.	RETI		– – – – – –
RETN	Return from a non maskable interrupt (NMI).	RETN		– – – – – –
RL r	Rotate bits in register r Left with Carry.	RL B	'r': (HL) (IX+#) (IY+#) A B C D E H L	S Z H P N C
RLC r	Rotate bits in register r Left and Copy the top bit to the Carry.	RLC B	'r': (HL) (IX+#) (IY+#) A B C D E H L	S Z H P N C
RLD	Rotate Left for binary coded Decimal.	RLD		S Z H V N –
RR r	Rotate bits in register r Right with carry.	RR B	'r': (HL) (IX+#) (IY+#) A B C D E H L	S Z H P N C
RRC r	Rotate bits in register r Right and Copy the bottom bit to the Carry.	RLC B	'r': (HL) (IX+#) (IY+#) A B C D E H L	S Z H P N C
RRD	Rotate Right for binary coded Decimal.	RRD		S Z H V N –
RST #	ReSeT function. RST is a single byte call to $00xx address.	RST $38		– – – – – –
SBC r	Subtract register r and the carry flag from the Accumulator A.	SBC B	'r': (HL) (IX+#) (IY+#) A B C D E H L IXH IXL IYH IYL	S Z H V N C
SBC A,#	Subtract 8 bit number # and the carry from A.	SBC 128	'#': 0-255 ($00-$FF)	S Z H V N C
SBC HL,rr	Subtract 16 bit register rr and the carry from HL.	SBC HL,BC	'rr': BC DE HL SP	S Z H V N C
SCF	Set the carry flag to 1.	SCF		– – H – N C
SET b,r	Set bit b from 8 bit register r to 1.	SET 7,B	'b': 0-7 (%76543210) 'r': (HL) (IX+#) (IY+#) A B C D E H L	– – – – – –
SLA r	Shift the bits register r Left for Arithmetic.	SLA A	'r': (HL) (IX+#) (IY+#) A B C D E H L	S Z H P N C
SLL r	Shift the bits in register r Left Logically (for unsigned numbers).	SLL A	'r': (HL) (IX+#) (IY+#) A B C D E H L	S Z H P N C
SRA r	Shift the bits in register r Right for Arithmetic. '	SRA A	'r': (HL) (IX+#) (IY+#) A B C D E H L	S Z H P N C
SRL r	Shift the bits in register r Right Logically.	SRL A	'r': (HL) (IX+#) (IY+#) A B C D E H L	S Z H P N C
SUB r	Subtract 8 bit register r from A.	SUB B	'r': (HL) (IX+#) (IY+#) A B C D E H L IXH IXL IYH IYL	S Z H V N C
SUB #	Subtract 8 bit value # from A.	SUB 32	'#': 0-255 ($00-$FF)	S Z H V N C
XOR r	Logical XOR (eXclusive OR) of bits in register r with Accumulator A.	XOR B	'r': (HL) (IX+#) (IY+#) A B C D E H L IXH IXL IYH IYL	S Z H V N C
XOR #	Logical XOR (eXclusive OR) of bits in immediate value # with Accumulator A.	XOR $64	'#': 0-255 ($00-$FF)	S Z H V N C

Condition	Meaning	Flag
Z	Zero	Z Set
NZ	Non Zero	Z Clear
C	Carry	C Set
NC	No Carry	C Clear
PO	Parity Odd	P/V Clear
PE	Parity Even	P/V Set
P	Positive Sign	S Clear
M	Minus Sign	S Set

Z80 - GBZ80

Instruction	Opcodes	B / T	Flags
ADC A,(HL)	8E	1 / 7	- z v c
ADC A,(IX+d)	DD 8E d	3 / 19	- z v c
ADC A,(IY+d)	FD 8E d	3 / 19	- z v c
ADC A,A	8F	1 / 4	- z v c
ADC A,B	88	1 / 4	- z v c
ADC A,C	89	1 / 4	- z v c
ADC A,D	8A	1 / 4	- z v c
ADC A,E	8B	1 / 4	- z v c
ADC A,H	8C	1 / 4	- z v c
ADC A,IXH	DD 8C	2 / 8	- z v c
ADC A,IYH	FD 8C	2 / 8	- z v c
ADC A,L	8D	1 / 4	- z v c
ADC A,IXL	DD 8D	2 / 8	- z v c
ADC A,IYL	FD 8D	2 / 8	- z v c
ADC A,n	CE n	2 / 7	- z v c
ADC HL,BC	ED 4A	2 / 15	- z v c
ADC HL,DE	ED 5A	2 / 15	- z v c
ADC HL,HL	ED 6A	2 / 15	- z v c
ADC HL,SP	ED 7A	2 / 15	- z v c
ADD A,(HL)	86	1 / 7	- z v c
ADD A,(IX+d)	DD 86 d	3 / 19	- z v c
ADD A,(IY+d)	FD 86 d	3 / 19	- z v c
ADD A,A	87	1 / 4	- z v c
ADD A,B	80	1 / 4	- z v c
ADD A,C	81	1 / 4	- z v c
ADD A,D	82	1 / 4	- z v c
ADD A,E	83	1 / 4	- z v c
ADD A,H	84	1 / 4	- z v c
ADD A,IXH	DD 84	2 / 8	- z v c
ADD A,IYH	FD 84	2 / 8	- z v c
ADD A,L	85	1 / 4	- z v c
ADD A,IXL	DD 85	2 / 8	- z v c
ADD A,IYL	FD 85	2 / 8	- z v c
ADD A,n	C6 n	2 / 7	- z v c
ADD HL,BC	09	1 / 11	- - - c
ADD HL,DE	19	1 / 11	- - - c
ADD HL,HL	29	1 / 11	- - - c
ADD HL,SP	39	1 / 11	- - - c
ADD IX,BC	DD 09	2 / 15	- - - c
ADD IX,DE	DD 19	2 / 15	- - - c
ADD IX,IX	DD 29	2 / 15	- - - c
ADD IX,SP	DD 39	2 / 15	- - - c
ADD IY,BC	FD 09	2 / 15	- - - c
ADD IY,DE	FD 19	2 / 15	- - - c
ADD IY,IY	FD 29	2 / 15	- - - c
ADD IY,SP	FD 39	2 / 15	- - - c
ADD SP,n	E8 n	1 / 16	n z v c
AND (HL)	A6	1 / 7	- z p c
AND (IX+d)	DD A6 d	3 / 19	- z p c
AND (IY+d)	FD A6 d	3 / 19	- z p c
AND A	A7	1 / 4	- z p c
AND B	A0	1 / 4	- z p c
AND C	A1	1 / 4	- z p c
AND D	A2	1 / 4	- z p c
AND E	A3	1 / 4	- z p c
AND H	A4	1 / 4	- z p c
AND IXH	DD A4	2 / 8	- z v c
AND IYH	FD A4	2 / 8	- z v c
AND L	A5	1 / 4	- z p c
AND IXL	DD A5	2 / 8	- z v c
AND IYL	FD A5	2 / 8	- z v c
AND n	E6 n	2 / 7	- z p c
BIT 0,(HL)	CB 46	2 / 12	? <>b ? -
BIT 0,(IX+d)	CB DD 46 d	4 / 20	? <>b ? -
BIT 0,(IY+d)	CB FD 46 d	4 / 20	? <>b ? -
BIT 0,A	CB 47	2 / 8	? <>b ? -
BIT 0,B	CB 40	2 / 8	? <>b ? -
BIT 0,C	CB 41	2 / 8	? <>b ? -
BIT 0,D	CB 42	2 / 8	? <>b ? -
BIT 0,E	CB 43	2 / 8	? <>b ? -
BIT 0,H	CB 44	2 / 8	? <>b ? -
BIT 0,L	CB 45	2 / 8	? <>b ? -
BIT 1,(HL)	CB 4E	2 / 12	? <>b ? -
BIT 1,(IX+d)	CB DD 4E d	4 / 20	? <>b ? -
BIT 1,(IY+d)	CB FD 4E d	4 / 20	? <>b ? -
BIT 1,A	CB 1F	2 / 8	? <>b ? -
BIT 1,B	CB 48	2 / 8	? <>b ? -
BIT 1,C	CB 49	2 / 8	? <>b ? -
BIT 1,D	CB 4A	2 / 8	? <>b ? -
BIT 1,E	CB 4B	2 / 8	? <>b ? -
BIT 1,H	CB 4C	2 / 8	? <>b ? -
BIT 1,L	CB 4D	2 / 8	? <>b ? -
BIT 2,(HL)	CB 56	2 / 12	? <>b ? -
BIT 2,(IY+d)	CB FD 56 d	4 / 20	? <>b ? -
BIT 2,(LY+d)	CB DD 56 d	4 / 20	? <>b ? -
BIT 2,A	CB 57	2 / 8	? <>b ? -
BIT 2,B	CB 50	2 / 8	? <>b ? -
BIT 2,C	CB 51	2 / 8	? <>b ? -
BIT 2,D	CB 52	2 / 8	? <>b ? -
BIT 2,E	CB 53	2 / 8	? <>b ? -
BIT 2,H	CB 54	2 / 8	? <>b ? -
BIT 2,L	CB 55	2 / 8	? <>b ? -
BIT 3,(HL)	CB 5E	2 / 12	? <>b ? -
BIT 3,(IX+d)	CB DD 5E d	4 / 20	? <>b ? -
BIT 3,(IY+d)	CB FD 5E d	4 / 20	? <>b ? -
BIT 3,A	CB 5F	2 / 8	? <>b ? -
BIT 3,B	CB 58	2 / 8	? <>b ? -
BIT 3,C	CB 59	2 / 8	? <>b ? -
BIT 3,D	CB 5A	2 / 8	? <>b ? -
BIT 3,E	CB 5B	2 / 8	? <>b ? -
BIT 3,H	CB 5C	2 / 8	? <>b ? -
BIT 3,L	CB 5D	2 / 8	? <>b ? -
BIT 4,(HL)	CB 66	2 / 12	? <>b ? -
BIT 4,(IY+d)	CB FD 66 d	4 / 20	? <>b ? -
BIT 4,(LY+d)	CB DD 66 d	4 / 20	? <>b ? -
BIT 4,A	CB 67	2 / 8	? <>b ? -
BIT 4,B	CB 60	2 / 8	? <>b ? -
BIT 4,C	CB 61	2 / 8	? <>b ? -
BIT 4,D	CB 62	2 / 8	? <>b ? -
BIT 4,E	CB 63	2 / 8	? <>b ? -
BIT 4,H	CB 64	2 / 8	? <>b ? -
BIT 4,L	CB 65	2 / 8	? <>b ? -
BIT 5,(HL)	CB 6E	2 / 12	? <>b ? -
BIT 5,(IX+d)	CB DD 6E d	4 / 20	? <>b ? -
BIT 5,(IY+d)	CB FD6E d	4 / 20	? <>b ? -
BIT 5,A	CB 6F	2 / 8	? <>b ? -
BIT 5,B	CB 68	2 / 8	? <>b ? -
BIT 5,C	CB 69	2 / 8	? <>b ? -
BIT 5,D	CB 6A	2 / 8	? <>b ? -
BIT 5,E	CB 6B	2 / 8	? <>b ? -
BIT 5,H	CB 6C	2 / 8	? <>b ? -
BIT 5,L	CB 6D	2 / 8	? <>b ? -
BIT 6,(HL)	CB 76	2 / 12	? <>b ? -
BIT 6,(IX+d)	CB DD 76 d	4 / 20	? <>b ? -
BIT 6,(IY+d)	CB FD 76 d	4 / 20	? <>b ? -
BIT 6,A	CB 77	2 / 8	? <>b ? -
BIT 6,B	CB 70	2 / 8	? <>b ? -
BIT 6,C	CB 71	2 / 8	? <>b ? -
BIT 6,D	CB 72	2 / 8	? <>b ? -
BIT 6,E	CB 73	2 / 8	? <>b ? -
BIT 6,H	CB 74	2 / 8	? <>b ? -
BIT 6,L	CB 75	2 / 8	? <>b ? -
BIT 7,(HL)	CB 7E	2 / 12	? <>b ? -
BIT 7,(IX+d)	CB DD 7E d	4 / 20	? <>b ? -
BIT 7,(IY+d)	CB FD 7E d	4 / 20	? <>b ? -
BIT 7,A	CB 7F	2 / 8	? <>b ? -
BIT 7,B	CB 78	2 / 8	? <>b ? -
BIT 7,C	CB 79	2 / 8	? <>b ? -
BIT 7,D	CB 7A	2 / 8	? <>b ? -
BIT 7,E	CB 7B	2 / 8	? <>b ? -
BIT 7,H	CB 7C	2 / 8	? <>b ? -
BIT 7,L	CB 7D	2 / 8	? <>b ? -
CALL addr	CD dr ad	3 / 17 20	- - - -
CALL c,addr	DC dr ad	3 / t17f10 12/20	- - - -
CALL m,addr	FC dr ad	3 / t17f10 12/20	- - - -
CALL nc,addr	D4 dr ad	3 / t17f10 12/20	- - - -
CALL nz,addr	C4 dr ad	3 / t17f10 12/20	- - - -
CALL p,addr	F4 dr ad	3 / t17f10 12/20	- - - -
CALL po,addr	E4 dr ad	3 / t17f10 12/20	- - - -
CALL pe,addr	EC dr ad	3 / t17f10 12/20	- - - -
CALL z,addr	CC dr ad	3 / t17f10 12/20	- - - -
CCF	3F	1 / 4	- - - x
CP (HL)	BE	1 / 7	- = v <
CP (IX+d)	DD BE d	3 / 19	- = v <
CP (IY+d)	FD BE d	3 / 19	- = v <
CP A	BF	1 / 4	- = v <
CP B	B8	1 / 4	- = v <
CP C	B9	1 / 4	- = v <
CP D	BA	1 / 4	- = v <
CP E	BB	1 / 4	- = v <
CP H	BC	1 / 4	- = v <
CP IXH	DD BC	2 / 8	- = v <
CP IYH	FD BC	2 / 8	- = v <
CP L	BD	1 / 4	- = v c
CP IXL	DD BD	2 / 8	- = v <
CP IYL	FD BD	2 / 8	- = v <
CP n	FE n	2 / 7	- = v <
CPD	ED A9	2 / 16	? - BC -
CPDR	ED B9	2 / =16#21	? - BC -
CPI	ED A1	2 / 16	? - BC -
CPIR	ED B2	2 / =16#21	? - BC -
CPL	2F	1 / 4	- - - -
DAA	27	1 / 4	- z p c
DEC (HL)	35	1 / 11	- z v -
DEC (IX+d)	DD 35d	3 / 23	- z v -
DEC (IY+d)	FD 35d	3 / 23	- z v -
DEC A	3D	1 / 4	- z v -
DEC B	5	1 / 4	- z v -
DEC BC	0B	1 / 6	- - - BUG
DEC C	0D	1 / 4	- z v -
DEC D	15	1 / 4	- z v -
DEC DE	1B	l / 6	- - - BUG
DEC E	1D	1 / 4	- z v -
DEC H	25	1 / 4	- z v -
DEC IXH	DD 25	2 / 8	- z v -
DEC IYH	FD 25	2 / 8	- z v -
DEC HL	2B	1 / 6	- - - BUG
DEC IX	DD 2B	2 / 10 12	- - - -
DEC IY	FD 2B	2 / 10 12	- - - -
DEC L	2D	1 / 4	- z v -
DEC IXL	DD 2D	2 / 8	- z v -
DEC IYL	FD 2D	2 / 8	- z v -
DEC SP	3B	1 / 6	- - - -
DI	F3	1 / 4	- - - -
DJNZ d	10 d	2 / t13f8	- - - -
EI	FB	1 / 4	- - - -
EX (SP),HL	E3	1 / 19	- - - -
EX (SP),IX	DD E3	2 / 23	- - - -
EX (SP),IY	FD E3	2 / 23	- - - -
EX AF,AF'	8	1 / 4	s' z' p' c'
EX DE,HL	EB	1 / 4	- - - -
EXX	D9	1 / 4	- - - -
HALT	76	1 / min 4	- - - BUG
IM 0	ED 46	2 / 8	- - - -
IM 1	ED 56	2 / 8	- - - -
IM 2	ED 5E	2 / 8	- - - -
IN A,(C)	ED 78	2 / 12 16	- z p -
IN A,(n)	DB n	2 / 11 12	- - - -
IN B,(C)	ED 40	2 / 12 16	- z p -
IN C,(C)	ED 48	2 / 12 16	- z p -
IN D,(C)	ED 50	2 / 12 16	- z p -
IN E,(C)	ED 58	2 / 12 16	- z p -
IN H,(C)	ED 60	2 / 12 16	- z p -
IN L,(C)	ED 68	2 / 12 16	- z p -
INC (HL)	34	1 / 11 12	- z v -
INC (IX+d)	DD 34 d	3 / 23 24	- z v -
INC (IY+d)	FD 34 d	3 / 23 24	- z v -
INC A	3C	1 / 4	- z v -
INC B	4	1 / 4	- z v -
INC BC	3	1 / 6 8	- - - BUG
INC C	0C	1 / 4	- z v -
INC D	14	1 / 4	- z v -
INC DE	13	1 / 6 8	- - - BUG
INC E	1C	1 / 4	- z v -
INC H	24	1 / 4	- z v -
INC IXH	DD 24	2 / 8	- z v -
INC IYH	FD 24	2 / 8	- z v -
INC HL	23	1 / 6 8	- - - BUG
INC IX	DD 23	2 / 10	- - - -
INC IY	FD 23	2 / 10	- - - -
INC L	2C	1 / 4	- z v -
INC IXL	DD 2C	2 / 8	- z v -
INC IYL	FD 2C	2 / 8	- z v -
INC SP	33	1 / 6	- - - -
IND	ED AA	2 / 16 25	? <>B ? -
INDR	ED BA	2 / z16 nz21	? - ? -
INI	ED A2	2 / 16 25	? <>B ? -
INIR	ED B2	2 / z16 nz21	? - ? -
JP (HL)	E9	1 / 4	- - - -
JP (IX)	DD E9	2 / 8	- - - -
JP (IY)	FD E9	2 / 8	- - - -
JP addr	C3 dr ad	3 / 10 12	- - - -
JP c,addr	DA dr ad	3 / 10 12	- - - -
JP m,addr	FA dr ad	3 / 10 12	- - - -
JP nc,addr	D2 dr ad	3 / 10 12	- - - -
JP nz,addr	C2 dr ad	3 / 10 12	- - - -
JP p,addr	F2 dr ad	3 / 10 12	- - - -
JP po,addr	E2 dr ad	3 / 10 12	- - - -
JP pe,addr	EA dr ad	3 / 10 12	- - - -
JP z,addr	CA dr ad	3 / 10 12	- - - -
JR c,d	38 d	2 / t12f7 8/12	- - - -
JR d	18 d	2 / 12	- - - -
JR nc,d	30 d	2 / t12f7 8/12	- - - -
JR nz,d	20 d	2 / t12f7 8/12	- - - -
JR z,d	28 d	2 / t12f7 8/12	- - - -
LD (addr),A	32 dr ad / EA dr ad	3 / 13	- - - -
LD (addr),BC	ED 43 dr ad	4 / 20 24	- - - -
LD (addr),DE	ED 53 dr ad	4 / 20 24	- - - -
LD (addr),HL	22 dr ad	3 / 16 24	- - - -
LD (addr),HL	ED 63 dr ad	4 / 20 24	- - - -
LD (addr),IX	DD 22 dr ad	4 / 20 24	- - - -
LD (addr),IY	FD 22 dr ad	4 / 20 24	- - - -
LD (addr),SP	ED 73 dr ad / 08 dr ad	4 / 20	- - - -
LD (BC),A	2	1 / 7	- - - -
LD (DE),A	12	1 / 7	- - - -
LD (HL),A	77	1 / 7	- - - -
LD (HL),B	70	1 / 7	- - - -
LD (HL),C	71	1 / 7	- - - -
LD (HL),D	72	1 / 7	- - - -
LD (HL),E	73	1 / 7	- - - -
LD (HL),H	74	1 / 7	- - - -
LD (HL),L	75	1 / 7	- - - -
LD (HL),n	36 n	2 / 10	- - - -
LD ($FF00+C),A	E2	2 / 8	- - - -
LD ($FF00+n) , A	E0 n	2 / 12	- - - -
LD (IX+d),A	DD 77 d	3 / 19	- - - -
LD (IX+d),B	DD 70 d	3 / 19	- - - -
LD (IX+d),C	DD 71 d	3 / 19	- - - -
LD (IX+d),D	DD 72 d	3 / 19	- - - -
LD (IX+d),E	DD 73 d	3 / 19	- - - -
LD (IX+d),H	DD 71 d	3 / 19	- - - -
LD (IX+d),L	DD 75 d	3 / 19	- - - -
LD (IX+d),n	DD 36 d n	4 / 19	- - - -
LD (IY+d),A	FD 77 d	3 / 19	- - - -
LD (IY+d),B	FD 70 d	3 / 19	- - - -
LD (IY+d),C	FD 71 d	3 / 19	- - - -
LD (IY+d),D	FD 72 d	3 / 19	- - - -
LD (IY+d),E	FD 73 d	3 / 19	- - - -
LD (IY+d),H	FD 74 d	3 / 19	- - - -
LD (IY+d),L	FD 75 d	3 / 19	- - - -
LD (IY+d),n	FD 36 d n	4 / 19	- - - -
LD A,(addr)	3A dr ad / F/A dr ad	3 / 16	- - - -
LD A,(BC)	0A	1 / 7 8	- - - -
LD A,(DE)	1A	1 / 7 8	- - - -
LD A,(HL)	7E	1 / 7 8	- - - -
LD A,(IX+d)	DD 7E d	3 / 19	- - - -
LD A,(IY+d)	FD 7E d	3 / 19	- - - -
LD A,A	7F	1 / 4	- - - -
LD A,B	78	1 / 4	- - - -
LD A,C	79	1 / 4	- - - -
LD A,D	7A	1 / 4	- - - -
LD A,E	7B	1 / 4	- - - -
LD A,H	7C	1 / 4	- - - -
LD A,IXH	DD 7C	2 / 8	- - - -
LD A,IYH	FD 7C	2 / 8	- - - -
LD A,I	ED 57	2 / 9	- z i -
LD A,L	7D	1 / 4	- - - -
LD A,IXL	DD 7D	2 / 8	- - - -
LD A,IYL	FD 7D	2 / 8	- - - -
LD A,n	3E n	2 / 7 8	- - - -
LD A, ($FF00+n)	F0 n	2 / 12	- - - -
LD A,($FF00+C)	F2	1 / 8	- - - -
LD A,R	ED 5F	2 / 9	- z i -
LD B,(HL)	46	1 / 7	- - - -
LD B,(IX+d)	DD 46 d	3 / 19	- - - -
LD B,(IY+d)	FD 46 d	3 / 19	- - - -
LD B,A	47	1 / 4	- - - -
LD B,B	40	1 / 4	- - - -
LD B,C	41	1 / 4	- - - -
LD B,D	42	1 / 4	- - - -
LD B,E	43	1 / 4	- - - -
LD B,H	44	1 / 4	- - - -
LD B,IXH	DD 44	2 / 8	- - - -
LD B,IYH	FD 44	2 / 8	- - - -
LD B,L	45	1 / 4	- - - -
LD B,IXL	DD 45	2 / 8	- - - -
LD B,IYL	FD 45	2 / 8	- - - -
LD B,n	06 n	2 / 7 8	- - - -
LD BC,(addr)	ED 4B dr ad	4 / 20	- - - -
LD BC,hilo	01 lo hi	3 / 10 12	- - - -
LD C,(HL)	4E	1 / 7	- - - -
LD C,(IX+d)	DD 4E d	3 / 19	- - - -
LD C,(IY+d)	DD 4E d	3 / 19	- - - -
LD C,A	4F	1 / 4	- - - -
LD C,B	48	1 / 4	- - - -
LD C,C	49	1 / 1	- - - -
LD C,D	4A	1 / 1	- - - -
LD C,E	4B	1 / 4	- - - -
LD C,H	4C	1 / 4	- - - -
LD C,IXH	DD 4C	2 / 8	- - - -
LD C,IYH	FD 4C	2 / 8	- - - -
LD C,L	4D	1 / 4	- - - -
LD C,IXL	DD 4D	2 / 8	- - - -
LD C,IYL	FD 4D	2 / 8	- - - -
LD C,n	0E n	2 / 7 8	- - - -
LD D,(HL)	56	1 / 7	- - - -
LD D,(IX+d)	DD 56 d	3 / 19	- - - -
LD D,(IY+d)	FD 56 d	3 / 19	- - - -
LD D,A	57	1 / 4	- - - -
LD D,B	50	1 / 4	- - - -
LD D,C	51	1 / 4	- - - -
LD D,D	52	1 / 4	- - - -
LD D,E	53	1 / 4	- - - -
LD D,H	54	1 / 4	- - - -
LD D,IXH	DD 54	2 / 8	- - - -
LD D,IYH	FD 54	2 / 8	- - - -
LD D,L	55	1 / 4	- - - -
LD D,IXL	DD 55	2 / 8	- - - -
LD D,IYL	FD 55	2 / 8	- - - -
LD D,n	16 n	2 / 7 8	- - - -
LD DE,(addr)	ED 5B dr ad	4 / 20	- - - -
LD DE,hilo	11 lo hi	3 / 10 12	- - - -
LD E,(HL)	5E	1 / 7	- - - -
LD E,(IX+d)	DD 5E d	3 / 19	- - - -
LD E,(IY+d)	FD 5E d	3 / 19	- - - -
LD E,A	5F	1 / 4	- - - -
LD E,B	58	1 / 4	- - - -
LD E,C	59	1 / 4	- - - -
LD E,D	5A	1 / 4	- - - -
LD E,E	5B	1 / 4	- - - -
LD E,H	5C	1 / 4	- - - -
LD E,IXH	DD 5C	2 / 8	- - - -
LD E,IYH	FD 5C	2 / 8	- - - -
LD E,L	5D	1 / 4	- - - -
LD E,IXL	DD 5D	2 / 8	- - - -
LD E,IYL	FD 5D	2 / 8	- - - -
LD E,n	1E n	2 / 7 8	- - - -
LD H,(HL)	66	1 / 7	- - - -
LD H,(IX+d)	DD 66 d	3 / 19	- - - -
LD H,(IY+d)	FD 66 d	3 / 19	- - - -
LD H,A	67	1 / 4	- - - -
LD H,B	60	1 / 4	- - - -
LD H,C	61	1 / 4	- - - -
LD H,D	62	1 / 4	- - - -
LD H,E	63	1 / 4	- - - -
LD H,H	64	1 / 4	- - - -
LD H,L	65	1 / 4	- - - -
LD H,n	26 n	2 / 7 8	- - - -
LD IXH,A	DD 67	2 / 8	- - - -
LD IXH,B	DD 60	2 / 8	- - - -
LD IXH,C	DD 61	2 / 8	- - - -
LD IXH,D	DD 62	2 / 8	- - - -
LD IXH,E	DD 63	2 / 8	- - - -
LD IXH,IXH	DD 64	2 / 8	- - - -
LD IXH,IXL	DD 65	2 / 8	- - - -
LD IXH,n	DD 26 n	3 / 11	- - - -
LD IYH,A	FD 67	2 / 8	- - - -
LD IYH,B	FD 60	2 / 8	- - - -
LD IYH,C	FD 61	2 / 8	- - - -
LD IYH,D	FD 62	2 / 8	- - - -
LD IYH,E	FD 63	2 / 8	- - - -
LD IYH,IYH	FD 64	2 / 8	- - - -
LD IYH,IYL	FD 65	2 / 8	- - - -
LD IYH,n	FD 26 n	3 / 11	- - - -
LD HL,(addr)	2A dr ad	3 / 16	- - - -
LD HL,(addr)	ED 6B dr ad	4 / 20	- - - -
LD HL,hilo	21 lo hi	3 / 10 12	- - - -
LD HL,SP+n	F8 n	2 / 12	- - - -
LD I,A	ED 47	2 / 9	- - - -
LD IX,(addr)	DD 2A dr ad	4 / 20	- - - -
LD IX,hilo	DD 21 lo hi	4 / 14	- - - -
LD IY,(addr)	FD 2A dr ad	4 / 20	- - - -
LD IY,hilo	FD 21 lo hi	4 / 14	- - - -
LD L,(HL)	6E	1 / 7	- - - -
LD L,(IX+d)	DD 6E d	3 / 19	- - - -
LD L,(IY+d)	FD 6E d	3 / 19	- - - -
LD L,A	6F	1 / 4	- - - -
LD L,B	68	1 / 4	- - - -
LD L,C	69	1 / 4	- - - -
LD L,D	6A	1 / 4	- - - -
LD L,E	6B	1 / 4	- - - -
LD L,H	6C	1 / 4	- - - -
LD L,L	6D	1 / 4	- - - -
LD L,n	2E n	2 / 7	- - - -
LD IXL,A	DD 6F	2 / 8	- - - -
LD IXL,B	DD 68	2 / 8	- - - -
LD IXL,C	DD 69	2 / 8	- - - -
LD IXL,D	DD 6A	2 / 8	- - - -
LD IXL,E	DD 6B	2 / 8	- - - -
LD IXL,IXH	DD 6C	2 / 8	- - - -
LD IXL,IXL	DD 6D	2 / 8	- - - -
LD IXL,n	DD 2E n	3 / 11	- - - -
LD IYL,A	FD 6F	2 / 8	- - - -
LD IYL,B	FD 68	2 / 8	- - - -
LD IYL,C	FD 69	2 / 8	- - - -
LD IYL,D	FD 6A	2 / 8	- - - -
LD IYL,E	FD 6B	2 / 8	- - - -
LD IYL,IYH	FD 6C	2 / 8	- - - -
LD IYL,IYL	FD 6D	2 / 8	- - - -
LD IYL,n	FD 2En	3 / 11	- - - -
LD R,A	ED 4F	2 / 9	- - - -
LD SP,(addr)	ED 7B dr ad	4 / 20	- - - -
LD SP,hilo	31 lo hi	3 / 10	- - - -
LD SP,HL	F9	1 / 6	- - - -
LD SP,IX	DD F9	2 / 10	- - - -
LD SP,IY	FD F9	2 / 10	- - - -
LDD	ED A8	2 / 16	- - BC -
LDDR	ED B8	2 / z16nz21	- - - -
LDI A,(HL)	2A	2 / 8	- - - -
LDI (HL),A	22	2 / 8	- - - -
LDD A,(HL)	32	2 / 8	- - - -
LDD (HL),A	3A	2 / 8	- - - -
LDI	ED A0	2 / 16	- - BC -
LDIR	ED B0	2 / zl6 nz21	- - - -
NEG	ED 44	2 / 8	- z A80 A0
NOP	0	1 / 4	- - - -
OR (HL)	B6	1 / 7	- z P -
OR (IX+d)	DD B6 d	3 / 19	- z p -
OR (IY+d)	FD B6 d	3 / 19	- z p -
OR A	B7	1 / 4	- z p -
OR B	B0	1 / 4	- z p -
OR C	B1	1 / 4	- z p -
OR D	B2	1 / 4	- z p -
OR E	B3	1 / 4	- z p -
OR H	B4	1 / 4	- z p -
OR IXH	DD B4	2 / 8	- z p -
OR IYH	FD B4	2 / 8	- z p -
OR L	B5	1 / 4	- z p -
OR IXL	DD B5	2 / 8	- z p -
OR IYL	FD B5	2 / 8	- z p -
OR n	F6 n	2 / 7	- z p -
OTDR	ED BB	2 / z16nz21	? - ? -
OTIR	ED B3	2 / zl6nz21	? l ? -
OUT (C),A	ED 79	2 / 12 16	- - - -
OUT (C),B	ED 41	2 / 12 16	- - - -
OUT (C),C	ED 49	2 / 12 16	- - - -
OUT (C),D	ED 51	2 / 12 16	- - - -
OUT (C),E	ED 59	2 / 12 16	- - - -
OUT (C),H	ED 61	2 / 12 16	- - - -
OUT (C),L	ED 69	2 / 12 16	- - - -
OUT (n),A	D3 n	2 / 11 12	- - - -
OUTD	ED AB	2 / 16 25	? <>B ? -
OUTI	ED A3	2 / 16 25	? <>B ? -
POP AF	F1	1 / 10	Popped
POP BC	C1	1 / 10	- - - -
POP DE	D1	1 / 10	- - - -
POP HL	E1	1 / 10	- - - -
POP IX	DD E1	2 / 14	- - - -
POP IY	FD E1	2 / 14	- - - -
PUSH AF	F5	1 / 11	- - - -
PUSH BC	C5	1 / 11	- - - -
PUSH DE	D5	1 / 11	- - - -
PUSH HL	E5	1 / 11	- - - -
PUSH IX	DD E5	2 / 15	- - - -
PUSH IY	FD E5	2 / 15	- - - -
RES 0,(HL)	CB 86	2 / 15	- - - -
RES 0,(IX+d)	DD CB d 86	4 / 23	- - - -
RES 0,(IY+d)	FD CB d 86	4 / 23	- - - -
RES 0,A	CB 87	2 / 8	- - - -
RES 0,B	CB 80	2 / 8	- - - -
RES 0,C	CB 81	2 / 8	- - - -
RES 0,D	CB 82	2 / 8	- - - -
RES 0,E	CB 83	2 / 8	- - - -
RES 0,H	CB 81	2 / 8	- - - -
RES 0,L	CB 85	2 / 8	- - - -
RES 1,(HL)	CB 8E	2 / 15	- - - -
RES 1,(IX+d)	DD CB d 8E	4 / 23	- - - -
RES 1,(IY+d)	FD CB d 8E	4 / 23	- - - -
RES 1,A	CB 8F	2 / 8	- - - -
RES 1,B	CB 88	2 / 8	- - - -
RES 1,C	CB 89	2 / 8	- - - -
RES 1,D	CB 8A	2 / 8	- - - -
RES 1,E	CB 8B	2 / 8	- - - -
RES 1,H	CB 8C	2 / 8	- - - -
RES 1,L	CB 8D	2 / 8	- - - -
RES 2,(HL)	CB 96	2 / 15	- - - -
RES 2,(IX+d)	DD CB d 96	4 / 23	- - - -
RES 2,(IY+d)	FD CB d 96	4 / 23	- - - -
RES 2,A	CB 97	2 / 8	- - - -
RES 2,B	CB 90	2 / 8	- - - -
RES 2,C	CB 91	2 / 8	- - - -
RES 2,D	CB 92	2 / 8	- - - -
RES 2,E	CB 93	2 / 8	- - - -
RES 2,H	CB 94	2 / 8	- - - -
RES 2,L	CB 95	2 / 8	- - - -
RES 3,(HL)	CB 9E	2 / 15	- - - -
RES 3,(IX+d)	DD CB d 9E	4 / 23	- - - -
RES 3,(IY+d)	FD CB d 9E	4 / 23	- - - -
RES 3,A	CB 9F	2 / 8	- - - -
RES 3,B	CB 98	2 / 8	- - - -
RES 3,C	CB 99	2 / 8	- - - -
RES 3,D	CB 9A	2 / 8	- - - -
RES 3,E	CB 9B	2 / 8	- - - -
RES 3,H	CB 9C	2 / 8	- - - -
RES 3,L	CB 9D	2 / 8	- - - -
RES 4,(HL)	CB A6	2 / 15	- - - -
RES 4,(IX+d)	DD CB d A6	4 / 23	- - - -
RES 4,(IY+d)	FD CB d A6	4 / 23	- - - -
RES 4,A	CB A7	2 / 8	- - - -
RES 4,B	CB A0	2 / 8	- - - -
RES 4,C	CB A1	2 / 8	- - - -
RES 4,D	CB A2	2 / 8	- - - -
RES 4,E	CB A3	2 / 8	- - - -
RES 4,H	CB A4	2 / 8	- - - -
RES 4,L	CB A5	2 / 8	- - - -
RES 5,(HL)	CB AE	2 / 15	- - - -
RES 5,(IX+d)	DD CB d AE	4 / 23	- - - -
RES 5,(IY+d)	FD CB d AE	4 / 23	- - - -
RES 5,A	CB AF	2 / 8	- - - -
RES 5,B	CB A8	2 / 8	- - - -
RES 5,C	CB A9	2 / 8	- - - -
RES 5,D	CB AA	2 / 8	- - - -
RES 5,E	CB AB	2 / 8	- - - -
RES 5,H	CB AC	2 / 8	- - - -
RES 5,L	CB AD	2 / 8	- - - -
RES 6,(HL)	CB B6	2 / 15	- - - -
RES 6,(IX+d)	DD CB d B6	4 / 23	- - - -
RES 6,(IY+d)	FD CB d B6	4 / 23	- - - -
RES 6,A	CB B7	2 / 8	- - - -
RES 6,B	CB B0	2 / 8	- - - -
RES 6,C	CB B1	2 / 8	- - - -
RES 6,D	CB B2	2 / 8	- - - -
RES 6,E	CB B3	2 / 8	- - - -
RES 6,H	CB B4	2 / 8	- - - -
RES 6,L	CB B5	2 / 8	- - - -
RES 7,(HL)	CB BE	2 / 15	- - - -
RES 7,(IX+d)	DD CB d BE	4 / 23	- - - -
RES 7,(IY+d)	FD CB d BE	4 / 23	- - - -
RES 7,A	CB BF	2 / 8	- - - -
RES 7,B	CB B8	2 / 8	- - - -
RES 7,C	CB B9	2 / 8	- - - -
RES 7,D	CB BA	2 / 8	- - - -
RES 7,E	CB BB	2 / 8	- - - -
RES 7,H	CB BC	2 / 8	- - - -
RES 7,L	CB BD	2 / 8	- - - -
RET	C9	1 / 10 12	- - - -
RET C	D8	1 / t11 f8 8/16	- - - -
RET M	F8	1 / t11 f8 8/16	- - - -
RET NC	D0	1 / t11 f8 8/16	- - - -
RET NZ	C0	1 / t11 f8 8/16	- - - -
RET P	F0	1 / t11 f8 8/16	- - - -
RET PE	E8	1 / t11 f8 8/16	- - - -
RET PO	E0	1 / t11 f8 8/16	- - - -
RET Z	C8	1 / t11 f8 8/16	- - - -
RETI	ED 4D / D9	1 / 8	- - - -
RETN	ED 45	2 / 14	- - - -
RL (HL)	CB 16	2 / 15	- z p r7
RL (IX+d)	DD CB d 16	4 / 23	- z p r7
RL (IY+d)	FD CB d 16	4 / 23	- z p r7
RL A	CB 17	2 / 8	- z p r7
RL B	CB 10	2 / 8	- z p r7
RL C	CB 11	2 / 8	- z p r7
RL D	CB 12	2 / 8	- z p r7
RL E	CB 13	2 / 8	- z p r7
RL H	CB 14	2 / 8	- z p r7
RL L	CB 15	2 / 8	- z p r7
RLA	17	1 / 4	- - - r7
RLC (HL)	CB 06	2 / 15	- z p r7
RLC (IX+d)	DD CB d 06	4 / 23	- z p r7
RLC (IY+d)	FD CB d 06	4 / 23	- z p r7
RLC A	CB 07	2 / 8	- z p r7
RLC B	CB 00	2 / 8	- z p r7
RLC C	CB 01	2 / 8	- z p r7
RLC D	CB 02	2 / 8	- z p r7
RLC E	CB 03	2 / 8	- z p r7
RLC H	CB 04	2 / 8	- z p r7
RLC L	CB 05	2 / 8	- z p r7
RLCA	7	1 / 4	- - - r7
RLD	ED 6F	2 / 18	- z p -
RR (HL)	CB 1E	2 / 15	- z p r0
RR (IX+d)	DD CB d 1E	4 / 23	- z p r0
RR (IY+d)	FD CB d 1E	4 / 23	- z p r0
RR A	CB 1F	2 / 8	- z p r0
RR B	CB 18	2 / 8	- z p r0
RR C	CB 19	2 / 8	- z p r0
RR D	CB 1A	2 / 8	- z p r0
RR E	CB 1B	2 / 8	- z p r0
RR H	CB 1C	2 / 8	- z p r0
RR L	CB 1D	2 / 8	- z p r0
RRA	1F	1 / 4	- - - r0
RRC (HL)	CB 0E	2 / 15	- z p r0
RRC (IX+d)	DD CB d 0E	4 / 23	- z p r0
RRC (IY+d)	FD CB d 0E	4 / 23	- z p r0
RRC A	CB 0F	2 / 8	- z p r0
RRC B	CB 08	2 / 8	- z p r0
RRC C	CB 09	2 / 8	- z p r0
RRC D	CB 0A	2 / 8	- z p r0
RRC E	CB 0B	2 / 8	- z p r0
RRC H	CB 0C	2 / 8	- z p r0
RRC L	CB 0D	2 / 8	- z p r0
RRCA	0F	1 / 4	- - - r0
RRD	ED 67	2 / 18	- z p -
RST 0	C7	1 / 11 16	- - - -
RST 8 (1)	CF	1 / 11 16	- - - -
RST 16 (2)	D7	1 / 11 16	- - - -
RST 24 (3)	DF	1 / 11 16	- - - -
RST 32 (4)	E7	1 / 11 16	- - - -
RST 40 (5)	EF	1 / 11 16	- - - -
RST 48 (6)	F7	1 / 11 16	- - - -
RST 56 (7)	FF	1 / 11 16	- - - -
SBC A,(HL)	9E	1 / 7	- z v b
SBC A,(IX+d)	DD 9Ed	3 / 19	- z v b
SBC A,(IY+d)	FD 9Ed	3 / 19	- z v b
SBC A,A	9F	1 / 4	- z v b
SBC A,B	98	1 / 4	- z v b
SBC A,C	99	1 / 4	- z v b
SBC A,D	9A	1 / 4	- z v b
SBC A,E	9B	1 / 4	- z v b
SBC A,H	9C	1 / 4	- z v b
SBC A,IXH	DD 9C	2 / 8	- z v b
SBC A,IYH	FD 9C	2 / 8	- z v b
SBC A,L	9D	1 / 4	- z v b
SBC A,IXL	DD 9D	2 / 8	- z v b
SBC A,IYL	FD 9D	2 / 8	- z v b
SBC A,n	DE n	2 / 7	- z v b
SBC HL,BC	ED 42	2 / 15	- z v b
SBC HL,DE	ED 52	2 / 15	- z v b
SBC HL,HL	ED 62	2 / 15	- z v b
SBC HL,SP	ED 72	2 / 15	- z v b
SCF	37	1 / 4	- - - -
SET 0,(HL)	CB C6	2 / 15	- - - -
SET 0,(IX+d)	DD CB d C6	4 / 23	- - - -
SET 0,(IY+d)	FD CB d C6	4 / 23	- - - -
SET 0,A	CB C7	2 / 8	- - - -
SET 0,B	CB C0	2 / 8	- - - -
SET 0,C	CB C1	2 / 8	- - - -
SET 0,D	CB C2	2 / 8	- - - -
SET 0,E	CB C3	2 / 8	- - - -
SET 0,H	CB C4	2 / 8	- - - -
SET 0,L	CB C5	2 / 8	- - - -
SET 1,(HL)	CB CE	2 / 15	- - - -
SET 1,(IX+d)	DD CB d CE	4 / 23	- - - -
SET 1,(IY+d)	FD CB d CE	4 / 23	- - - -
SET 1,A	CB CF	2 / 8	- - - -
SET 1,B	CB C8	2 / 8	- - - -
SET 1,C	CB C9	2 / 8	- - - -
SET 1,D	CB CA	2 / 8	- - - -
SET 1,E	CB CB	2 / 8	- - - -
SET 1,H	CB CC	2 / 8	- - - -
SET 1,L	CB CD	2 / 8	- - - -
SET 2,(HL)	CB D6	2 / 15	- - - -
SET 2,(IX+d)	DD CB d D6	4 / 23	- - - -
SET 2,(IY+d)	FD CB d D6	4 / 23	- - - -
SET 2,A	CB D7	2 / 8	- - - -
SET 2,B	CB D0	2 / 8	- - - -
SET 2,C	CB D1	2 / 8	- - - -
SET 2,D	CB D2	2 / 8	- - - -
SET 2,E	CB D3	2 / 8	- - - -
SET 2,H	CB D4	2 / 8	- - - -
SET 2,L	CB D5	2 / 8	- - - -
SET 3,(HL)	CB DE	2 / 15	- - - -
SET 3,(IX+d)	DD CB d DE	4 / 23	- - - -
SET 3,(IY+d)	FD CB d DE	4 / 23	- - - -
SET 3,A	CB DF	2 / 8	- - - -
SET 3,B	CB D8	2 / 8	- - - -
SET 3,C	CB D9	2 / 8	- - - -
SET 3,D	CB DA	2 / 8	- - - -
SET 3,E	CB DB	2 / 8	- - - -
SET 3,H	CB DC	2 / 8	- - - -
SET 3,L	CB DD	2 / 8	- - - -
SET 4,(HL)	CB E6	2 / 15	- - - -
SET 4,(IX+d)	DD CB d E6	4 / 23	- - - -
SET 4,(IY+d)	FD CB d E6	4 / 23	- - - -
SET 4,A	CB E7	2 / 8	- - - -
SET 4,B	CB E0	2 / 8	- - - -
SET 4,C	CB E1	2 / 8	- - - -
SET 4,D	CB E2	2 / 8	- - - -
SET 4,E	CB E3	2 / 8	- - - -
SET 4,H	CB E4	2 / 8	- - - -
SET 4,L	CB E5	2 / 8	- - - -
SET 5,(HL)	CB EE	2 / 15	- - - -
SET 5,(IX+d)	DD CB d EE	4 / 23	- - - -
SET 5,(IY+d)	FD CB d EE	4 / 23	- - - -
SET 5,A	CB EF	2 / 8	- - - -
SET 5,B	CB E8	2 / 8	- - - -
SET 5,C	CB E9	2 / 8	- - - -
SET 5,D	CB EA	2 / 8	- - - -
SET 5,E	CB EB	2 / 8	- - - -
SET 5,H	CB EC	2 / 8	- - - -
SET 5,L	CB ED	2 / 8	- - - -
SET 6,(HL)	CB F6	2 / 15	- - - -
SET 6,(IX+d)	DD CB d F6	4 / 23	- - - -
SET 6,(IY+d)	FD CB d F6	4 / 23	- - - -
SET 6,A	CB F7	2 / 8	- - - -
SET 6,B	CB F0	2 / 8	- - - -
SET 6,C	CB F1	2 / 8	- - - -
SET 6,D	CB F2	2 / 8	- - - -
SET 6,E	CB F3	2 / 8	- - - -
SET 6,H	CB F4	2 / 8	- - - -
SET 6,L	CB F5	2 / 8	- - - -
SET 7,(HL)	CB FE	2 / 15	- - - -
SET 7,(IX+d)	DD CB d FE	4 / 23	- - - -
SET 7,(IY+d)	FD CB d FE	4 / 23	- - - -
SET 7,A	CB FF	2 / 8	- - - -
SET 7,B	CB F8	2 / 8	- - - -
SET 7,C	CB F9	2 / 8	- - - -
SET 7,D	CB FA	2 / 8	- - - -
SET 7,E	CB FB	2 / 8	- - - -
SET 7,H	CB FC	2 / 8	- - - -
SET 7,L	CB FD	2 / 8	- - - -
SLA (HL)	CB 26	2 / 15	- z p r7
SLA (IX+d)	DD CB d 26	4 / 23	- z p r7
SLA (IY+d)	FD CB d 26	4 / 23	- z p r7
SLA A	CB 27	2 / 8	- z p r7
SLA B	CB 20	2 / 8	- z p r7
SLA C	CB 21	2 / 8	- z p r7
SLA D	CB 22	2 / 8	- z p r7
SLA E	CB 23	2 / 8	- z p r7
SLA H	CB 24	2 / 8	- z p r7
SLA L	CB 25	2 / 8	- z p r7
SLL (HL)	CB 36	2 / 15	- z p r7
SLL (IX+d)	DD CB d 36	4 / 23	- z p r7
SLL (IY+d)	FD CB d 36	4 / 23	- z p r7
SLL A	CB 37	2 / 8	- z p r7
SLL B	CB 30	2 / 8	- z p r7
SLL C	CB 31	2 / 8	- z p r7
SLL D	CB 32	2 / 8	- z p r7
SLL E	CB 33	2 / 8	- z p r7
SLL H	CB 34	2 / 8	- z p r7
SLL L	CB 35	2 / 8	- z p r7
SRA (HL)	CB 2E	2 / 15	- z p r0
SRA (IX+d)	DD CB d 2E	4 / 23	- z p r0
SRA (IY+d)	FD CB d 2E	4 / 23	- z p r0
SRA A	CB 2F	2 / 8	- z p r0
SRA B	CB 28	2 / 8	- z p r0
SRA C	CB 29	2 / 8	- z p r0
SRA D	CB 2A	2 / 8	- z p r0
SRA E	CB 2B	2 / 8	- z p r0
SRA H	CB 2C	2 / 8	- z p r0
SRA L	CB 2D	2 / 8	- z p r0
SRL (HL)	CB 3E	2 / 15	- z p r0
SRL (IX+d)	DD CB d 3E	4 / 23	- z p r0
SRL (IY+d)	FD CB d 3E	4 / 23	- z p r0
SRL A	CB 3F	2 / 8	- z p r0
SRL B	CB 38	2 / 8	- z p r0
SRL C	CB 39	2 / 8	- z p r0
SRL D	CB 3A	2 / 8	- z p r0
SRL E	CB 3B	2 / 8	- z p r0
SRL H	CB 3C	2 / 8	- z p r0
SRL L	CB 3D	2 / 8	- z p r0
STOP	10 00	2 / 4	- - - -
SUB (HL)	96	1 / 7	- z v b
SUB (IX+d)	DD 96 d	3 / 19	- z v b
SUB (IY+d)	FD 96 d	3 / 19	- z v b
SUB A	97	1 / 4	- z v b
SUB B	90	1 / 4	- z v b
SUB C	91	1 / 4	- z v b
SUB D	92	1 / 4	- z v b
SUB E	93	1 / 4	- z v b
SUB H	94	1 / 4	- z v b
SUB IXH	DD AC	2 / 8	- z v b
SUB IYH	FD AC	2 / 8	- z v b
SUB L	95	1 / 4	- z v b
SUB IXL	DD AD	2 / 8	- z v b
SUB IYL	FD AD	2 / 8	- z v b
SUB n	D6 n	2 / 7	- z v b
SWAP A	CB 37	2 / 8	- z v z
SWAP B	CB 30	2 / 8	- z v z
SWAP C	CB 31	2 / 8	- z v z
SWAP D	CB 32	2 / 8	- z v z
SWAP E	CB 33	2 / 8	- z v z
SWAP H	CB 34	2 / 8	- z v z
SWAP L	CB 35	2 / 8	- z v z
SWAP (HL)	CB 36	2 / 16	- z v z
XOR (HL)	AE	1 / 7	- z p -
XOR (IX+d)	DD AC d	3 / 19	- z p -
XOR (IY+d)	FD AC d	3 / 19	- z p -
XOR A	AF	1 / 4	- z p -
XOR B	A8	1 / 4	- z p -
XOR C	A9	1 / 4	- z p -
XOR D	AA	1 / 4	- z p -
XOR E	AB	1 / 4	- z p -
XOR H	AC	1 / 4	- z p -
XOR IXH	DD AC	2 / 8	- z p -
XOR IYH	FD AD	2 / 8	- z p -
XOR L	AD	1 / 4	- z p -
XOR IXL	DD AC	2 / 8	- z p -
XOR IYL	FD AD	2 / 8	- z p -
XOR n	EE n	2 / 7	- z p -

Black: GBZ80 & Z80
Blue: Z80 only
Red: GBZ80 only
Purple: Cpc Timings

Alternate instructions with the same meaning:

LD ($FF00+n) , A	LDH (n),A
LD A, ($FF00+n)	LDH A,(n)
LDI (HL),A	LD (HLI),A or LD (HL+),A
LDI A,(HL)	LD A,(HLI) or LD A,(HL+)
LDD (HL),A	LD (HLD),A or LD (HL-),A
LDD A,(HL)	LD A,(HL-) or LD A,(HLD)
LD A,(C)	LD A,($FF00+C)
LD (C),A	LD ($FF00+C),A
LD HL,SP+n	LDHL SP,n ..

GB BUG: INC rr / DEC rr when rr is between &FE00-&FEFF will cause sprites to flicker

GB BUG: HALT with DI set will not stop the CPU, but will 'skip' one command, so put a NOP after HALT

eZ80

Instruction	Opcodes	B / T	Flags
ADC A,(HL)	8E	1 / 2	s z h p n c
ADC A,(IX+d)	DD 8E d	3 / 4	s z h p n c
ADC A,(IY+d)	FD 8E d	3 / 4	s z h p n c
ADC A,A	8F	1 / 1	s z h p n c
ADC A,B	88	1 / 4	s z h p n c
ADC A,C	89	1 / 4	s z h p n c
ADC A,D	8A	1 / 4	s z h p n c
ADC A,E	8B	1 / 4	s z h p n c
ADC A,H	8C	1 / 4	s z h p n c
ADC A,IXH	DD 8C	2 / 2	s z h p n c
ADC A,IXL	DD 8D	2 / 2	s z h p n c
ADC A,IYH	FD 8C	2 / 2	s z h p n c
ADC A,IYL	FD 8D	2 / 2	s z h p n c
ADC A,L	8D	1 / 4	s z h p n c
ADC A,n	CE n	2 / 2	s z h p n c
ADC HL,BC	ED 4A	2 / 2	s z h p n c
ADC HL,DE	ED 5A	2 / 2	s z h p n c
ADC HL,HL	ED 6A	2 / 2	s z h p n c
ADC HL,SP	ED 7A	2 / 2	s z h p n c
ADD A,(HL)	86	1 / 2	s z h p n c
ADD A,(IX+d)	DD 86 d	3 / 4	s z h p n c
ADD A,(IY+d)	FD 86 d	3 / 4	s z h p n c
ADD A,A	87	1 / 1	s z h p n c
ADD A,B	80	1 / 1	s z h p n c
ADD A,C	81	1 / 1	s z h p n c
ADD A,D	82	1 / 1	s z h p n c
ADD A,E	83	1 / 1	s z h p n c
ADD A,H	84	1 / 1	s z h p n c
ADD A,IXH	DD 84	2 / 2	s z h p n c
ADD A,IXL	DD 85	2 / 2	s z h p n c
ADD A,IYH	FD 84	2 / 2	s z h p n c
ADD A,IYL	FD 85	2 / 2	s z h p n c
ADD A,L	85	1 / 1	s z h p n c
ADD A,n	C6 n	2 / 2	s z h p n c
ADD HL,BC	09	1 / 1	- - h - n c
ADD HL,DE	19	1 / 1	- - h - n c
ADD HL,HL	29	1 / 1	- - h - n c
ADD HL,SP	39	1 / 1	- - h - n c
ADD IX,BC	DD 09	2 / 2	- - h - n c
ADD IX,DE	DD 19	2 / 2	- - h - n c
ADD IX,IX	DD 29	2 / 2	- - h - n c
ADD IX,SP	DD 39	2 / 2	- - h - n c
ADD IY,BC	FD 09	2 / 2	- - h - n c
ADD IY,DE	FD 19	2 / 2	- - h - n c
ADD IY,IY	FD 29	2 / 2	- - h - n c
ADD IY,SP	FD 39	2 / 2	- - h - n c
AND (HL)	A6	1 / 2	s z h p n c
AND (IX+d)	DD A6 d	2 / 3	s z h p n c
AND (IY+d)	FD A6 d	2 / 3	s z h p n c
AND A	A7	1 / 1	s z h p n c
AND B	A0	1 / 1	s z h p n c
AND C	A1	1 / 1	s z h p n c
AND D	A2	1 / 1	s z h p n c
AND E	A3	1 / 1	s z h p n c
AND H	A4	1 / 1	s z h p n c
AND IXH	DD A4	2 / 2	s z h p n c
AND IXL	DD A5	2 / 2	s z h p n c
AND IYH	FD A4	2 / 2	s z h p n c
AND IYL	FD A5	2 / 2	s z h p n c
AND L	A5	1 / 1	s z h p n c
AND n	E6 n	2 / 2	s z h p n c
BIT 0,(HL)	CB 46	2 / 3	s z h p n -
BIT 0,(IX+d)	CB DD 46 d	4 / 5	s z h p n -
BIT 0,(IY+d)	CB FD 46 d	4 / 5	s z h p n -
BIT 0,A	CB 47	2 / 2	s z h p n -
BIT 0,B	CB 40	2 / 2	s z h p n -
BIT 0,C	CB 41	2 / 2	s z h p n -
BIT 0,D	CB 42	2 / 2	s z h p n -
BIT 0,E	CB 43	2 / 2	s z h p n -
BIT 0,H	CB 44	2 / 2	s z h p n -
BIT 0,L	CB 45	2 / 2	s z h p n -
BIT 1,(HL)	CB 4E	2 / 3	s z h p n -
BIT 1,(IX+d)	CB DD 4E d	4 / 5	s z h p n -
BIT 1,(IY+d)	CB FD 4E d	4 / 5	s z h p n -
BIT 1,A	CB 1F	2 / 2	s z h p n -
BIT 1,B	CB 48	2 / 2	s z h p n -
BIT 1,C	CB 49	2 / 2	s z h p n -
BIT 1,D	CB 4A	2 / 2	s z h p n -
BIT 1,E	CB 4B	2 / 2	s z h p n -
BIT 1,H	CB 4C	2 / 2	s z h p n -
BIT 1,L	CB 4D	2 / 2	s z h p n -
BIT 2,(HL)	CB 56	2 / 3	s z h p n -
BIT 2,(IX+d)	CB DD 56 d	4 / 5	s z h p n -
BIT 2,(IY+d)	CB FD 56 d	4 / 5	s z h p n -
BIT 2,A	CB 57	2 / 2	s z h p n -
BIT 2,B	CB 50	2 / 2	s z h p n -
BIT 2,C	CB 51	2 / 2	s z h p n -
BIT 2,D	CB 52	2 / 2	s z h p n -
BIT 2,E	CB 53	2 / 2	s z h p n -
BIT 2,H	CB 54	2 / 2	s z h p n -
BIT 2,L	CB 55	2 / 2	s z h p n -
BIT 3,(HL)	CB 5E	2 / 3	s z h p n -
BIT 3,(IX+d)	CB DD 5E d	4 / 5	s z h p n -
BIT 3,(IY+d)	CB FD 5E d	4 / 5	s z h p n -
BIT 3,A	CB 5F	2 / 2	s z h p n -
BIT 3,B	CB 58	2 / 2	s z h p n -
BIT 3,C	CB 59	2 / 2	s z h p n -
BIT 3,D	CB 5A	2 / 2	s z h p n -
BIT 3,E	CB 5B	2 / 2	s z h p n -
BIT 3,H	CB 5C	2 / 2	s z h p n -
BIT 3,L	CB 5D	2 / 2	s z h p n -
BIT 4,(HL)	CB 66	2 / 3	s z h p n -
BIT 4,(IX+d)	CB DD 66 d	4 / 5	s z h p n -
BIT 4,(IY+d)	CB FD 66 d	4 / 5	s z h p n -
BIT 4,A	CB 67	2 / 2	s z h p n -
BIT 4,B	CB 60	2 / 2	s z h p n -
BIT 4,C	CB 61	2 / 2	s z h p n -
BIT 4,D	CB 62	2 / 2	s z h p n -
BIT 4,E	CB 63	2 / 2	s z h p n -
BIT 4,H	CB 64	2 / 2	s z h p n -
BIT 4,L	CB 65	2 / 2	s z h p n -
BIT 5,(HL)	CB 6E	2 / 3	s z h p n -
BIT 5,(IX+d)	CB DD 6E d	4 / 5	s z h p n -
BIT 5,(IY+d)	CB FD 6E d	4 / 5	s z h p n -
BIT 5,A	CB 6F	2 / 2	s z h p n -
BIT 5,B	CB 68	2 / 2	s z h p n -
BIT 5,C	CB 69	2 / 2	s z h p n -
BIT 5,D	CB 6A	2 / 2	s z h p n -
BIT 5,E	CB 6B	2 / 2	s z h p n -
BIT 5,H	CB 6C	2 / 2	s z h p n -
BIT 5,L	CB 6D	2 / 2	s z h p n -
BIT 6,(HL)	CB 76	2 / 3	s z h p n -
BIT 6,(IX+d)	CB DD 76 d	4 / 5	s z h p n -
BIT 6,(IY+d)	CB FD 76 d	4 / 5	s z h p n -
BIT 6,A	CB 77	2 / 2	s z h p n -
BIT 6,B	CB 70	2 / 2	s z h p n -
BIT 6,C	CB 71	2 / 2	s z h p n -
BIT 6,D	CB 72	2 / 2	s z h p n -
BIT 6,E	CB 73	2 / 2	s z h p n -
BIT 6,H	CB 74	2 / 2	s z h p n -
BIT 6,L	CB 75	2 / 2	s z h p n -
BIT 7,(HL)	CB 7E	2 / 3	s z h p n -
BIT 7,(IX+d)	CB DD 7E d	4 / 5	s z h p n -
BIT 7,(IY+d)	CB FD 7E d	4 / 5	s z h p n -
BIT 7,A	CB 7F	2 / 2	s z h p n -
BIT 7,B	CB 78	2 / 2	s z h p n -
BIT 7,C	CB 79	2 / 2	s z h p n -
BIT 7,D	CB 7A	2 / 2	s z h p n -
BIT 7,E	CB 7B	2 / 2	s z h p n -
BIT 7,H	CB 7C	2 / 2	s z h p n -
BIT 7,L	CB 7D	2 / 2	s z h p n -
CALL addr	CD dr ad	3 / 5	- - - - - -
CALL c,addr	DC dr ad	3 / 6+	- - - - - -
CALL m,addr	FC dr ad	3 / 6+	- - - - - -
CALL nc,addr	D4 dr ad	3 / 6+	- - - - - -
CALL nz,addr	C4 dr ad	3 / 6+	- - - - - -
CALL p,addr	F4 dr ad	3 / 6+	- - - - - -
CALL pe,addr	EC dr ad	3 / 6+	- - - - - -
CALL po,addr	E4 dr ad	3 / 6+	- - - - - -
CALL z,addr	CC dr ad	3 / 6+	- - - - - -
CCF	3F	1 / 1	- - h - n c
CP (HL)	BE	1 / 2	s z h p n c
CP (IX+d)	DD BE d	3 / 4	s z h p n c
CP (IY+d)	FD BE d	3 / 19	s z h p n c
CP A	BF	1 / 4	s z h p n c
CP B	B8	1 / 4	s z h p n c
CP C	B9	1 / 4	s z h p n c
CP D	BA	1 / 4	s z h p n c
CP E	BB	1 / 4	s z h p n c
CP H	BC	1 / 4	s z h p n c
CP IXH	DD BC	2 / 2	s z h p n c
CP IXL	DD BD	2 / 2	s z h p n c
CP IYH	FD BC	2 / 2	s z h p n c
CP IYL	FD BD	2 / 2	s z h p n c
CP L	BD	1 / 4	s z h p n c
CP n	FE n	2 / 2	s z h p n c
CPD	ED A9	2 / 3	s z h p n -
CPDR	ED B9	?	s z h p n -
CPI	ED A1	2 / 3	s z h p n -
CPIR	ED B1	?	s z h p n -
CPL	2F	1 / 1	- - h - n -
DAA	27	1 / 1	s z h p - c
DEC (HL)	35	1 / 4	s z h p n -
DEC (IX+d)	DD 35 d	3 / 6	s z h p n -
DEC (IY+d)	FD 35 d	3 / 6	s z h p n -
DEC A	3D	1 / 1	s z h p n -
DEC B	05	1 / 4	s z h p n -
DEC BC	0B	1 / 1	- - - - - -
DEC C	0D	1 / 4	s z h p n -
DEC D	15	1 / 4	s z h p n -
DEC DE	1B	1 / 1	- - - - - -
DEC E	1D	1 / 4	s z h p n -
DEC H	25	1 / 4	s z h p n -
DEC HL	2B	1 / 1	- - - - - -
DEC IX	DD 2B	2 / 2	- - - - - -
DEC IXH	DD 25	2 / 2	s z h p n -
DEC IXL	DD 2D	2 / 2	s z h p n -
DEC IY	FD 2B	2 / 2	- - - - - -
DEC IYH	FD 25	2 / 2	s z h p n -
DEC IYL	FD 2D	2 / 2	s z h p n -
DEC L	2D	1 / 4	s z h p n -
DEC SP	3B	1 / 1	- - - - - -
DI	F3	1 / 1	- - - - - -
DJNZ d	10 d	2 / 2-4	- - - - - -
EI	FB	1 / 1	- - - - - -
EX (SP),HL	E3	1 / 5-7	- - - - - -
EX (SP),IX	DD E3	2 / 6-8	- - - - - -
EX (SP),IY	FD E3	2 / 6-8	- - - - - -
EX AF,AF'	08	1 / 4	s z h p n c
EX DE,HL	EB	1 / 1	- - - - - -
EXX	D9	1 / 1	- - - - - -
HALT	76	1 / 1	- - - - - -
IM 0	ED 46	1 / 2	- - - - - -
IM 1	ED 56	1 / 2	- - - - - -
IM 2	ED 5E	1 / 2	- - - - - -
IN A,(BC)	ED 78	2 / 3	s z h p n -
IN A,(n)	DB n	2 / 3	- - - - - -
IN B,(BC)	ED 40	2 / 3	s z h p n -
IN C,(BC)	ED 48	2 / 3	s z h p n -
IN D,(BC)	ED 50	2 / 3	s z h p n -
IN E,(BC)	ED 58	2 / 3	s z h p n -
IN H,(BC)	ED 60	2 / 3	s z h p n -
IN L,(BC)	ED 68	2 / 3	s z h p n -
IN0 A,(n)	ED 38	2 / 4	s z h p n -
IN0 B,(n)	ED 00	2 / 4	s z h p n -
IN0 C,(n)	ED 08	2 / 4	s z h p n -
IN0 D,(n)	ED 10	2 / 4	s z h p n -
IN0 E,(n)	ED 18	2 / 4	s z h p n -
IN0 H,(n)	ED 20	2 / 4	s z h p n -
IN0 L,(n)	ED 28	2 / 4	s z h p n -
INC (HL)	34	1 / 4	s z h p n -
INC (IX+d)	DD 34 d	3 / 6	s z h p n -
INC (IY+d)	FD 34 d	3 / 6	s z h p n -
INC A	3C	1 / 1	s z h p n -
INC B	04	1 / 1	s z h p n -
INC BC	03	1 / 1	- - - - - -
INC C	0C	1 / 1	s z h p n -
INC D	14	1 / 1	s z h p n -
INC DE	13	1 / 1	- - - - - -
INC E	1C	1 / 1	s z h p n -
INC H	24	1 / 1	s z h p n -
INC HL	23	1 / 1	- - - - - -
INC IX	DD 23	2 / 2	- - - - - -
INC IXH	DD 24	2 / 2	s z h p n -
INC IXL	DD 2C	2 / 2	s z h p n -
INC IY	FD 23	2 / 2	- - - - - -
INC IYH	FD 24	2 / 2	s z h p n -
INC IYL	FD 2C	2 / 2	s z h p n -
INC L	2C	1 / 1	s z h p n -
INC SP	33	1 / 1	- - - - - -
IND	ED AA	2 / 5	- z - - n -
IND2	ED 8C	2 / 5	- z - - n -
IND2R	ED 9C	2 / ?	- z - - n -
INDM	ED 8A	2 / 5	- z - - n -
INDMR	ED 9A	2 / ?	- z - - n -
INDR	ED BA	?	- z - - n -
INDRX	ED CA	2 / ?	- z - - n -
INI	ED A2	5	- z - - n -
INI2	ED 84	2 / 5	- z - - n -
INI2R	ED 94	2 / ?	- z - - n -
INIM	ED 82	2 / 5	- z - - n -
INIMR	ED 92	2 / ?	- z - - n -
INIR	ED B2	2 / ?	- z - - n -
INIRX	ED C2	2 / ?	- z - - n -
JP (HL)	E9	1 / 3	- - - - - -
JP (IX)	DD E9	2 / 4	- - - - - -
JP (IY)	FD E9	2 / 4	- - - - - -
JP addr	C3 dr ad	2 / 4+	- - - - - -
JP c,addr	DA dr ad	3 / 3+	- - - - - -
JP m,addr	FA dr ad	3 / 3+	- - - - - -
JP nc,addr	D2 dr ad	3 / 3+	- - - - - -
JP nz,addr	C2 dr ad	3 / 3+	- - - - - -
JP p,addr	F2 dr ad	3 / 3+	- - - - - -
JP pe,addr	EA dr ad	3 / 3+	- - - - - -
JP po,addr	E2 dr ad	3 / 3+	- - - - - -
JP z,addr	CA dr ad	3 / 3+	- - - - - -
JR c,d	38 d	2 / 2+	- - - - - -
JR d	18 d	2 / 3	- - - - - -
JR nc,d	30 d	2 / 2+	- - - - - -
JR nz,d	20 d	2 / 2+	- - - - - -
JR z,d	28 d	2 / 2+	- - - - - -
LD (addr),A	32 dr ad	3 / 4	- - - - - -
LD (addr),BC	ED 43 dr ad	4 / 6	- - - - - -
LD (addr),DE	ED 53 dr ad	4 / 6	- - - - - -
LD (addr),HL	22 dr ad	3 / 5	- - - - - -
LD (addr),IX	DD 22 dr ad	4 / 6	- - - - - -
LD (addr),IY	FD 22 dr ad	4 / 6	- - - - - -
LD (addr),SP	ED 73 dr ad	4 / 6	- - - - - -
LD (BC),A	02	1 / 2	- - - - - -
LD (DE),A	12	1 / 2	- - - - - -
LD (HL),A	77	1 / 2	- - - - - -
LD (HL),B	70	1 / 2	- - - - - -
LD (HL),BC	ED 0F	2 / 4+	- - - - - -
LD (HL),C	71	1 / 2	- - - - - -
LD (HL),D	72	1 / 2	- - - - - -
LD (HL),DE	ED 1F	2 / 4+	- - - - - -
LD (HL),E	73	1 / 2	- - - - - -
LD (HL),H	74	1 / 2	- - - - - -
LD (HL),HL	ED 2F	2 / 4+	- - - - - -
LD (HL),IX	ED 3F	2 / 4+	- - - - - -
LD (HL),IY	ED 3E	2 / 4+	- - - - - -
LD (HL),L	75	1 / 2	- - - - - -
LD (HL),n	36 n	2 / 3	- - - - - -
LD (IX+d),A	DD 77 d	3 / 4	- - - - - -
LD (IX+d),B	DD 70 d	3 / 4	- - - - - -
LD (IX+d),BC	DD 0F d	3 / 5+	- - - - - -
LD (IX+d),C	DD 71 d	3 / 4	- - - - - -
LD (IX+d),D	DD 72 d	3 / 4	- - - - - -
LD (IX+d),DE	DD 1F d	3 / 5+	- - - - - -
LD (IX+d),E	DD 73 d	3 / 4	- - - - - -
LD (IX+d),H	DD 71 d	3 / 4	- - - - - -
LD (IX+d),HL	DD 2F d	3 / 5+	- - - - - -
LD (IX+d),IX	DD 3F d	3 / 5+	- - - - - -
LD (IX+d),IY	DD 3E d	3 / 5+	- - - - - -
LD (IX+d),L	DD 75 d	3 / 4	- - - - - -
LD (IX+d),n	DD 36 d n	4 / 5	- - - - - -
LD (IY+d),A	FD 77 d	3 / 4	- - - - - -
LD (IY+d),B	FD 70 d	3 / 4	- - - - - -
LD (IY+d),BC	FD 0F d	3 / 5+	- - - - - -
LD (IY+d),C	FD 71 d	3 / 4	- - - - - -
LD (IY+d),D	FD 72 d	3 / 4	- - - - - -
LD (IY+d),DE	FD 1F d	3 / 5+	- - - - - -
LD (IY+d),E	FD 73 d	3 / 4	- - - - - -
LD (IY+d),H	FD 74 d	3 / 4	- - - - - -
LD (IY+d),HL	FD 2F d	3 / 5+	- - - - - -
LD (IY+d),IX	FD 3F d	3 / 5+	- - - - - -
LD (IY+d),IY	FD 3E d	3 / 5+	- - - - - -
LD (IY+d),L	FD 75 d	3 / 4	- - - - - -
LD (IY+d),n	FD 36 d n	4 / 5	- - - - - -
LD A,(addr)	3A dr ad	3 / 4+	- - - - - -
LD A,(BC)	0A	1 / 2	- - - - - -
LD A,(DE)	1A	1 / 2	- - - - - -
LD A,(HL)	7E	1 / 2	- - - - - -
LD A,(IX+d)	DD 7E d	3 / 4	- - - - - -
LD A,(IY+d)	FD 7E d	3 / 4	- - - - - -
LD A,A	7F	1 / 1	- - - - - -
LD A,B	78	1 / 1	- - - - - -
LD A,C	79	1 / 1	- - - - - -
LD A,D	7A	1 / 1	- - - - - -
LD A,E	7B	1 / 1	- - - - - -
LD A,H	7C	1 / 1	- - - - - -
LD A,I	ED 57	2 / 2	s z h p n -
LD A,IXH	DD 7C	2 / 2	- - - - - -
LD A,IXL	DD 7D	2 / 2	- - - - - -
LD A,IYH	FD 7C	2 / 2	- - - - - -
LD A,IYL	FD 7D	2 / 2	- - - - - -
LD A,L	7D	1 / 1	- - - - - -
LD A,MB	ED 6E	2 / 2	- - - - - -
LD A,n	3E n	2 / 2	- - - - - -
LD A,R	ED 5F	2 / 2	s z h p n -
LD B,(HL)	46	1 / 2	- - - - - -
LD B,(IX+d)	DD 46 d	3 / 4	- - - - - -
LD B,(IY+d)	FD 46 d	3 / 4	- - - - - -
LD B,A	47	1 / 1	- - - - - -
LD B,C	41	1 / 1	- - - - - -
LD B,D	42	1 / 1	- - - - - -
LD B,E	43	1 / 1	- - - - - -
LD B,H	44	1 / 1	- - - - - -
LD B,IXH	DD 44	2 / 2	- - - - - -
LD B,IXL	DD 45	2 / 2	- - - - - -
LD B,IYH	FD 44	2 / 2	- - - - - -
LD B,IYL	FD 45	2 / 2	- - - - - -
LD B,L	45	1 / 1	- - - - - -
LD B,n	06 n	2 / 2	- - - - - -
LD BC,(addr)	ED 4B dr ad	4 / 6	- - - - - -
LD BC,(HL)	ED 07	2 / 4+	- - - - - -
LD BC,(IX+d)	DD 07 d	3 / 5+	- - - - - -
LD BC,(IY+d)	FD 07 d	3 / 5+	- - - - - -
LD BC,hilo	01 lo hi	3 / 3	- - - - - -
LD C,(HL)	4E	1 / 2	- - - - - -
LD C,(IX+d)	DD 4E d	3 / 4	- - - - - -
LD C,(IY+d)	DD 4E d	3 / 4	- - - - - -
LD C,A	4F	1 / 1	- - - - - -
LD C,B	48	1 / 1	- - - - - -
LD C,D	4A	1 / 1	- - - - - -
LD C,E	4B	1 / 1	- - - - - -
LD C,H	4C	1 / 1	- - - - - -
LD C,IXH	DD 4C	2 / 2	- - - - - -
LD C,IXL	DD 4D	2 / 2	- - - - - -
LD C,IYH	FD 4C	2 / 2	- - - - - -
LD C,IYL	FD 4D	2 / 2	- - - - - -
LD C,L	4D	1 / 1	- - - - - -
LD C,n	0E n	2 / 2	- - - - - -
LD D,(HL)	56	1 / 2	- - - - - -
LD D,(IX+d)	DD 56 d	3 / 4	- - - - - -
LD D,(IY+d)	FD 56 d	3 / 4	- - - - - -
LD D,A	57	1 / 1	- - - - - -
LD D,B	50	1 / 1	- - - - - -
LD D,C	51	1 / 1	- - - - - -
LD D,E	53	1 / 1	- - - - - -
LD D,H	54	1 / 1	- - - - - -
LD D,IXH	DD 54	2 / 2	- - - - - -
LD D,IXL	DD 55	2 / 2	- - - - - -
LD D,IYH	FD 54	2 / 2	- - - - - -
LD D,IYL	FD 55	2 / 2	- - - - - -
LD D,L	55	1 / 1	- - - - - -
LD D,n	16 n	2 / 2	- - - - - -
LD DE,(addr)	ED 5B dr ad	4 / 6	- - - - - -
LD DE,(HL)	ED 17	2 / 4+	- - - - - -
LD DE,(IX+d)	DD 17 d	3 / 5+	- - - - - -
LD DE,(IY+d)	FD 17 d	3 / 5+	- - - - - -
LD DE,hilo	11 lo hi	3 / 3	- - - - - -
LD E,(HL)	5E	1 / 2	- - - - - -
LD E,(IX+d)	DD 5E d	3 / 4	- - - - - -
LD E,(IY+d)	FD 5E d	3 / 4	- - - - - -
LD E,A	5F	1 / 1	- - - - - -
LD E,B	58	1 / 1	- - - - - -
LD E,C	59	1 / 1	- - - - - -
LD E,D	5A	1 / 1	- - - - - -
LD E,H	5C	1 / 1	- - - - - -
LD E,IXH	DD 5C	2 / 2	- - - - - -
LD E,IXL	DD 5D	2 / 2	- - - - - -
LD E,IYH	FD 5C	2 / 2	- - - - - -
LD E,IYL	FD 5D	2 / 2	- - - - - -
LD E,L	5D	1 / 4	
LD E,n	1E n	2 / 2	- - - - - -
LD H,(HL)	66	1 / 2	- - - - - -
LD H,(IX+d)	DD 66 d	3 / 4	- - - - - -
LD H,(IY+d)	FD 66 d	3 / 4	- - - - - -
LD H,A	67	1 / 1	- - - - - -
LD H,B	60	1 / 1	- - - - - -
LD H,C	61	1 / 1	- - - - - -
LD H,D	62	1 / 1	- - - - - -
LD H,E	63	1 / 1	- - - - - -
LD H,H	64	1 / 1	- - - - - -
LD H,L	65	1 / 1	- - - - - -
LD H,n	26 n	2 / 2	- - - - - -
LD HL,(addr)	2A dr ad	3 / 5	- - - - - -
LD HL,(HL)	ED 27	2 / 4+	- - - - - -
LD HL,(IX+d)	DD 27 d	3 / 5+	- - - - - -
LD HL,(IY+d)	FD 27 d	3 / 5+	- - - - - -
LD HL,hilo	21 lo hi	3 / 3	- - - - - -
LD HL,I	ED D7	2 / 2	s z h p n -
LD I,A	ED 47	2 / 2	- - - - - -
LD I,HL	ED C7	2 / 2	- - - - - -
LD IX,(addr)	DD 2A dr ad	4 / 6	- - - - - -
LD IX,(HL)	ED 37	2 / 4+	- - - - - -
LD IX,(IX+d)	DD 37 d	3 / 5+	- - - - - -
LD IX,(IY+d)	FD 37 d	3 / 5+	- - - - - -
LD IX,hilo	DD 21 lo hi	4 / 4+	- - - - - -
LD IXH,A	DD 67	2 / 2	- - - - - -
LD IXH,B	DD 60	2 / 2	- - - - - -
LD IXH,C	DD 61	2 / 2	- - - - - -
LD IXH,D	DD 62	2 / 2	- - - - - -
LD IXH,E	DD 63	2 / 2	- - - - - -
LD IXH,IXH	DD 64	2 / 2	- - - - - -
LD IXH,IXL	DD 65	2 / 2	- - - - - -
LD IXH,n	DD 26 n	2 / 2	- - - - - -
LD IXL,A	DD 6F	2 / 2	- - - - - -
LD IXL,B	DD 68	2 / 2	- - - - - -
LD IXL,C	DD 69	2 / 2	- - - - - -
LD IXL,D	DD 6A	2 / 2	- - - - - -
LD IXL,E	DD 6B	2 / 2	- - - - - -
LD IXL,IXH	DD 6C	2 / 2	- - - - - -
LD IXL,IXL	DD 6D	2 / 2	- - - - - -
LD IXL,n	DD 2E n	2 / 2	- - - - - -
LD IY,(addr)	FD 2A dr ad	4 / 6	- - - - - -
LD IY,(HL)	ED 31	2 / 4+	- - - - - -
LD IY,(IX+d)	DD 31 d	3 / 5+	- - - - - -
LD IY,(IX+d)	FD 31 d	3 / 5+	- - - - - -
LD IY,hilo	FD 21 lo hi	4 / 4+	- - - - - -
LD IYH,A	FD 67	2 / 2	- - - - - -
LD IYH,B	FD 60	2 / 2	- - - - - -
LD IYH,C	FD 61	2 / 2	- - - - - -
LD IYH,D	FD 62	2 / 2	- - - - - -
LD IYH,E	FD 63	2 / 2	- - - - - -
LD IYH,IYH	FD 64	2 / 2	- - - - - -
LD IYH,IYL	FD 65	2 / 2	- - - - - -
LD IYH,n	FD 26 n	2 / 2	- - - - - -
LD IYL,A	FD 6F	2 / 2	- - - - - -
LD IYL,B	FD 68	2 / 2	- - - - - -
LD IYL,C	FD 69	2 / 2	- - - - - -
LD IYL,D	FD 6A	2 / 2	- - - - - -
LD IYL,E	FD 6B	2 / 2	- - - - - -
LD IYL,IYH	FD 6C	2 / 2	- - - - - -
LD IYL,IYL	FD 6D	2 / 2	- - - - - -
LD IYL,n	FD 2E n	2 / 2	- - - - - -
LD L,(HL)	6E	1 / 2	- - - - - -
LD L,(IX+d)	DD 6E d	3 / 4	- - - - - -
LD L,(IY+d)	FD 6E d	3 / 4	- - - - - -
LD L,A	6F	1 / 1	- - - - - -
LD L,B	68	1 / 1	- - - - - -
LD L,C	69	1 / 1	- - - - - -
LD L,D	6A	1 / 1	- - - - - -
LD L,E	6B	1 / 1	- - - - - -
LD L,H	6C	1 / 1	- - - - - -
LD L,L	6D	1 / 1	- - - - - -
LD L,n	2E n	2 / 2	- - - - - -
LD MB,A	ED 6D	2 / 2	- - - - - -
LD R,A	ED 4F	2 / 2	- - - - - -
LD SP,(addr)	ED 7B dr ad	4 / 5	- - - - - -
LD SP,hilo	31 lo hi	3 / 3	3 / 3
LD SP,HL	F9	1 / 1	- - - - - -
LD SP,IX	DD F9	2 / 2	- - - - - -
LD SP,IY	FD F9	2 / 2	- - - - - -
LDD	ED A8	2 / 5	- - h p n -
LDDR	ED B8	2 / ?	- - h p n -
LDI	ED A0	2 / 5	- - h p n -
LDIR	ED B0	2 / ?	- - h p n -
LEA BC,IX+d	ED 02 d	3 / 3	- - - - - -
LEA BC,IY+d	ED 03 d	3 / 3	- - - - - -
LEA DE,IX+d	ED 12 d	3 / 3	- - - - - -
LEA DE,IY+d	ED 13 d	3 / 3	- - - - - -
LEA HL,IX+d	ED 22 d	3 / 3	- - - - - -
LEA HL,IY+d	ED 23 d	3 / 3	- - - - - -
LEA IX,IX+d	ED 32 d	3 / 3	- - - - - -
LEA IX,IY+d	ED 54 d	3 / 3	- - - - - -
LEA IY,IX+d	ED 55 d	3 / 3	- - - - - -
LEA IY,IY+d	ED 33 d	3 / 3	- - - - - -
MLT BC	ED 4C	2 / 6	- - - - - -
MLT DE	ED 5C	2 / 6	- - - - - -
MLT HL	ED 6C	2 / 6	- - - - - -
MLT SP	ED 7C	2 / 6	- - - - - -
NEG	ED 44	2 / 2	s z h p n c
NOP	00	1 / 1	- - - - - -
OR (HL)	B6	1 / 2	s z h p n c
OR (IX+d)	DD B6 d	3 / 4	- - - - - -
OR (IY+d)	FD B6 d	3 / 4	- - - - - -
OR A	B7	1 / 1	s z h p n c
OR B	B0	1 / 1	s z h p n c
OR C	B1	1 / 1	s z h p n c
OR D	B2	1 / 1	s z h p n c
OR E	B3	1 / 1	s z h p n c
OR H	B4	1 / 1	s z h p n c
OR IXH	DD B4	2 / 2	s z h p n c
OR IXL	DD B5	2 / 2	s z h p n c
OR IYH	FD B4	2 / 2	s z h p n c
OR IYL	FD B5	2 / 2	s z h p n c
OR L	B5	1 / 1	s z h p n c
OR n	F6 n	2 / 2	s z h p n c
OTD2R	ED BC	2 / ?	- z - - n -
OTDM	ED 8B	2 / 5	- z - - n -
OTDMR	ED 9B	2 / ?	- z - - n -
OTDR	ED BB	2 / ?	- z - - n -
OTDRX	ED CB	2 / ?	- z - - n -
OTI2R	ED B4	2 / ?	- z - - n -
OTIM	ED 83	2 / 5	- z - - n -
OTIMR	ED 93	2 / ?	- z - - n -
OTIR	ED B3	2 / ?	- z - - n -
OTIRX	ED C3	2 / ?	- z - - n -
OUT (BC),A	ED 79	2 / 3	- - - - - -
OUT (BC),B	ED 41	2 / 3	- - - - - -
OUT (BC),C	ED 49	2 / 3	- - - - - -
OUT (BC),D	ED 51	2 / 3	- - - - - -
OUT (BC),E	ED 59	2 / 3	- - - - - -
OUT (BC),H	ED 61	2 / 3	- - - - - -
OUT (BC),L	ED 69	2 / 3	- - - - - -
OUT (n),A	D3 n	2 / 3	- - - - - -
OUT0 (n),A	ED 39 n	3 / 4	- - - - - -
OUT0 (n),B	ED 01 n	3 / 4	- - - - - -
OUT0 (n),C	ED 09 n	3 / 4	- - - - - -
OUT0 (n),D	ED 11 n	3 / 4	- - - - - -
OUT0 (n),E	ED 19 n	3 / 4	- - - - - -
OUT0 (n),H	ED 21 n	3 / 4	- - - - - -
OUT0 (n),L	ED 29 n	3 / 4	- - - - - -
OUTD	ED AB	2 / 5	- z - - n -
OUTD2	ED AC	2 / 5	- z - - n -
OUTI	ED A3	2 / 5	- z - - n -
OUTI2	ED A4	2 / 5	- z - - n -
PEA IX+d	ED 65 d	3/5+	- - - - - -
PEA IY+d	ED 66 d	3/5+	- - - - - -
POP AF	F1	1 / 3+	Popped
POP BC	C1	1 / 3+	- - - - - -
POP DE	D1	1 / 3+	- - - - - -
POP HL	E1	1 / 3+	- - - - - -
POP IX	DD E1	2/4+	- - - - - -
POP IY	FD E1	2/4+	- - - - - -
PUSH AF	F5	1 / 3+	- - - - - -
PUSH BC	C5	1 / 3+	- - - - - -
PUSH DE	D5	1 / 3+	- - - - - -
PUSH HL	E5	1 / 3+	- - - - - -
PUSH IX	DD E5	2 / 4+	- - - - - -
PUSH IY	FD E5	2 / 4+	- - - - - -
RES 0,(HL)	CB 86	2 / 3	- - - - - -
RES 0,(IX+d)	DD CB d 86	4 / 5	- - - - - -
RES 0,(IY+d)	FD CB d 86	4 / 5	- - - - - -
RES 0,A	CB 87	2 / 2	- - - - - -
RES 0,B	CB 80	2 / 2	- - - - - -
RES 0,C	CB 81	2 / 2	- - - - - -
RES 0,D	CB 82	2 / 2	- - - - - -
RES 0,E	CB 83	2 / 2	- - - - - -
RES 0,H	CB 81	2 / 2	- - - - - -
RES 0,L	CB 85	2 / 2	- - - - - -
RES 1,(HL)	CB 8E	2 / 3	- - - - - -
RES 1,(IX+d)	DD CB d 8E	4 / 5	- - - - - -
RES 1,(IY+d)	FD CB d 8E	4 / 5	- - - - - -
RES 1,A	CB 8F	2 / 2	- - - - - -
RES 1,B	CB 88	2 / 2	- - - - - -
RES 1,C	CB 89	2 / 2	- - - - - -
RES 1,D	CB 8A	2 / 2	- - - - - -
RES 1,E	CB 8B	2 / 2	- - - - - -
RES 1,H	CB 8C	2 / 2	- - - - - -
RES 1,L	CB 8D	2 / 2	- - - - - -
RES 2,(HL)	CB 96	2 / 3	- - - - - -
RES 2,(IX+d)	DD CB d 96	4 / 5	- - - - - -
RES 2,(IY+d)	FD CB d 96	4 / 5	- - - - - -
RES 2,A	CB 97	2 / 2	- - - - - -
RES 2,B	CB 90	2 / 2	- - - - - -
RES 2,C	CB 91	2 / 2	- - - - - -
RES 2,D	CB 92	2 / 2	- - - - - -
RES 2,E	CB 93	2 / 2	- - - - - -
RES 2,H	CB 94	2 / 2	- - - - - -
RES 2,L	CB 95	2 / 2	- - - - - -
RES 3,(HL)	CB 9E	2 / 3	- - - - - -
RES 3,(IX+d)	DD CB d 9E	4 / 5	- - - - - -
RES 3,(IY+d)	FD CB d 9E	4 / 5	- - - - - -
RES 3,A	CB 9F	2 / 2	- - - - - -
RES 3,B	CB 98	2 / 2	- - - - - -
RES 3,C	CB 99	2 / 2	- - - - - -
RES 3,D	CB 9A	2 / 2	- - - - - -
RES 3,E	CB 9B	2 / 2	- - - - - -
RES 3,H	CB 9C	2 / 2	- - - - - -
RES 3,L	CB 9D	2 / 2	- - - - - -
RES 4,(HL)	CB A6	2 / 3	- - - - - -
RES 4,(IX+d)	DD CB d A6	4 / 5	- - - - - -
RES 4,(IY+d)	FD CB d A6	4 / 5	- - - - - -
RES 4,A	CB A7	2 / 2	- - - - - -
RES 4,B	CB A0	2 / 2	- - - - - -
RES 4,C	CB A1	2 / 2	- - - - - -
RES 4,D	CB A2	2 / 2	- - - - - -
RES 4,E	CB A3	2 / 2	- - - - - -
RES 4,H	CB A4	2 / 2	- - - - - -
RES 4,L	CB A5	2 / 2	- - - - - -
RES 5,(HL)	CB AE	2 / 3	- - - - - -
RES 5,(IX+d)	DD CB d AE	4 / 5	- - - - - -
RES 5,(IY+d)	FD CB d AE	4 / 5	- - - - - -
RES 5,A	CB AF	2 / 2	- - - - - -
RES 5,B	CB A8	2 / 2	- - - - - -
RES 5,C	CB A9	2 / 2	- - - - - -
RES 5,D	CB AA	2 / 2	- - - - - -
RES 5,E	CB AB	2 / 2	- - - - - -
RES 5,H	CB AC	2 / 2	- - - - - -
RES 5,L	CB AD	2 / 2	- - - - - -
RES 6,(HL)	CB B6	2 / 3	- - - - - -
RES 6,(IX+d)	DD CB d B6	4 / 5	- - - - - -
RES 6,(IY+d)	FD CB d B6	4 / 5	- - - - - -
RES 6,A	CB B7	2 / 2	- - - - - -
RES 6,B	CB B0	2 / 2	- - - - - -
RES 6,C	CB B1	2 / 2	- - - - - -
RES 6,D	CB B2	2 / 2	- - - - - -
RES 6,E	CB B3	2 / 2	- - - - - -
RES 6,H	CB B4	2 / 2	- - - - - -
RES 6,L	CB B5	2 / 2	- - - - - -
RES 7,(HL)	CB BE	2 / 3	- - - - - -
RES 7,(IX+d)	DD CB d BE	4 / 5	- - - - - -
RES 7,(IY+d)	FD CB d BE	4 / 5	- - - - - -
RES 7,A	CB BF	2 / 2	- - - - - -
RES 7,B	CB B8	2 / 2	- - - - - -
RES 7,C	CB B9	2 / 2	- - - - - -
RES 7,D	CB BA	2 / 2	- - - - - -
RES 7,E	CB BB	2 / 2	- - - - - -
RES 7,H	CB BC	2 / 2	- - - - - -
RES 7,L	CB BD	2 / 2	- - - - - -
RET	C9	1 / 5+	- - - - - -
RET C	D8	1 / 2+	- - - - - -
RET M	F8	1 / 2+	- - - - - -
RET NC	D0	1 / 2+	- - - - - -
RET NZ	C0	1 / 2+	- - - - - -
RET P	F0	1 / 2+	- - - - - -
RET PE	E8	1 / 2+	- - - - - -
RET PO	E0	1 / 2+	- - - - - -
RET Z	C8	1 / 2+	- - - - - -
RETI	ED 4D	2 / 6+	- - - - - -
RETN	ED 45	2 / 6+	- - - - - -
RL (HL)	CB 16	2 / 5	s z h p n c
RL (IX+d)	DD CB d 16	4 / 7	s z h p n c
RL (IY+d)	FD CB d 16	4 / 7	s z h p n c
RL A	CB 17	2 / 2	s z h p n c
RL B	CB 10	2 / 2	s z h p n c
RL C	CB 11	2 / 2	s z h p n c
RL D	CB 12	2 / 2	s z h p n c
RL E	CB 13	2 / 2	s z h p n c
RL H	CB 14	2 / 2	s z h p n c
RL L	CB 15	2 / 2	s z h p n c
RLA	17	1 / 1	- - h - n c
RLC (HL)	CB 06	2 / 5	s z h p n c
RLC (IX+d)	DD CB d 06	4 / 7	s z h p n c
RLC (IY+d)	FD CB d 06	4 / 7	s z h p n c
RLC A	CB 07	2 / 2	s z h p n c
RLC B	CB 00	2 / 2	s z h p n c
RLC C	CB 01	2 / 2	s z h p n c
RLC D	CB 02	2 / 2	s z h p n c
RLC E	CB 03	2 / 2	s z h p n c
RLC H	CB 04	2 / 2	s z h p n c
RLC L	CB 05	2 / 2	s z h p n c
RLCA	07	1 / 4	- - h - n c
RLD	ED 6F	2 / 5	s z h p n c
RR (HL)	CB 1E	2 / 5	s z h p n c
RR (IX+d)	DD CB d 1E	4 / 7	s z h p n c
RR (IY+d)	FD CB d 1E	4 / 7	s z h p n c
RR A	CB 1F	2 / 2	s z h p n c
RR B	CB 18	2 / 2	s z h p n c
RR C	CB 19	2 / 2	s z h p n c
RR D	CB 1A	2 / 2	s z h p n c
RR E	CB 1B	2 / 2	s z h p n c
RR H	CB 1C	2 / 2	s z h p n c
RR L	CB 1D	2 / 2	s z h p n c
RRA	1F	1 / 1	- - h - n c
RRC (HL)	CB 0E	1 / 5	s z h p n c
RRC (IX+d)	DD CB d 0E	4 / 7	s z h p n c
RRC (IY+d)	FD CB d 0E	4 / 23	s z h p n c
RRC A	CB 0F	2 / 2	s z h p n c
RRC B	CB 08	2 / 2	s z h p n c
RRC C	CB 09	2 / 2	s z h p n c
RRC D	CB 0A	2 / 2	s z h p n c
RRC E	CB 0B	2 / 2	s z h p n c
RRC H	CB 0C	2 / 2	s z h p n c
RRC L	CB 0D	2 / 2	s z h p n c
RRCA	0F	1 / 1	- - h - n c
RRD	ED 67	2 / 5	s z h p n -
RSMIX	ED 7E	2 / 2	- - - - - -
RST 0	C7	1 / 5+	- - - - - -
RST 16 (2)	D7	1 / 5+	- - - - - -
RST 24 (3)	DF	1 / 5+	- - - - - -
RST 32 (4)	E7	1 / 5+	- - - - - -
RST 40 (5)	EF	1 / 5+	- - - - - -
RST 48 (6)	F7	1 / 5+	- - - - - -
RST 56 (7)	FF	1 / 5+	- - - - - -
RST 8 (1)	CF	1 / 5+	- - - - - -
SBC A,(HL)	9E	1 / 2	s z h p n c
SBC A,(IX+d)	DD 9Ed	3 / 4	s z h p n c
SBC A,(IY+d)	FD 9Ed	3 / 4	s z h p n c
SBC A,A	9F	1 / 1	s z h p n c
SBC A,B	98	1 / 1	s z h p n c
SBC A,C	99	1 / 1	s z h p n c
SBC A,D	9A	1 / 1	s z h p n c
SBC A,E	9B	1 / 1	s z h p n c
SBC A,H	9C	1 / 1	s z h p n c
SBC A,IXH	DD 9C	2 / 2	s z h p n c
SBC A,IXL	DD 9D	2 / 2	s z h p n c
SBC A,IYH	FD 9C	2 / 2	s z h p n c
SBC A,IYL	FD 9D	2 / 2	s z h p n c
SBC A,L	9D	1 / 1	s z h p n c
SBC A,n	DE n	2 / 2	s z h p n c
SBC HL,BC	ED 42	2 / 2	s z h p n c
SBC HL,DE	ED 52	2 / 2	s z h p n c
SBC HL,HL	ED 62	2 / 2	s z h p n c
SBC HL,SP	ED 72	2 / 2	s z h p n c
SCF	37	1 / 1	- - h - n c
SET 0,(HL)	CB C6	2 / 3	- - - - - -
SET 0,(IX+d)	DD CB d C6	4 / 5	- - - - - -
SET 0,(IY+d)	FD CB d C6	4 / 5	- - - - - -
SET 0,A	CB C7	2 / 2	- - - - - -
SET 0,B	CB C0	2 / 2	- - - - - -
SET 0,C	CB C1	2 / 2	- - - - - -
SET 0,D	CB C2	2 / 2	- - - - - -
SET 0,E	CB C3	2 / 2	- - - - - -
SET 0,H	CB C4	2 / 2	- - - - - -
SET 0,L	CB C5	2 / 2	- - - - - -
SET 1,(HL)	CB CE	2 / 3	- - - - - -
SET 1,(IX+d)	DD CB d CE	4 / 5	- - - - - -
SET 1,(IY+d)	FD CB d CE	4 / 5	- - - - - -
SET 1,A	CB CF	2 / 2	- - - - - -
SET 1,B	CB C8	2 / 2	- - - - - -
SET 1,C	CB C9	2 / 2	- - - - - -
SET 1,D	CB CA	2 / 2	- - - - - -
SET 1,E	CB CB	2 / 2	- - - - - -
SET 1,H	CB CC	2 / 2	- - - - - -
SET 1,L	CB CD	2 / 2	- - - - - -
SET 2,(HL)	CB D6	2 / 3	- - - - - -
SET 2,(IX+d)	DD CB d D6	4 / 5	- - - - - -
SET 2,(IY+d)	FD CB d D6	4 / 5	- - - - - -
SET 2,A	CB D7	2 / 2	- - - - - -
SET 2,B	CB D0	2 / 2	- - - - - -
SET 2,C	CB D1	2 / 2	- - - - - -
SET 2,D	CB D2	2 / 2	- - - - - -
SET 2,E	CB D3	2 / 2	- - - - - -
SET 2,H	CB D4	2 / 2	- - - - - -
SET 2,L	CB D5	2 / 2	- - - - - -
SET 3,(HL)	CB DE	2 / 3	- - - - - -
SET 3,(IX+d)	DD CB d DE	4 / 5	- - - - - -
SET 3,(IY+d)	FD CB d DE	4 / 5	- - - - - -
SET 3,A	CB DF	2 / 2	- - - - - -
SET 3,B	CB D8	2 / 2	- - - - - -
SET 3,C	CB D9	2 / 2	- - - - - -
SET 3,D	CB DA	2 / 2	- - - - - -
SET 3,E	CB DB	2 / 2	- - - - - -
SET 3,H	CB DC	2 / 2	- - - - - -
SET 3,L	CB DD	2 / 2	- - - - - -
SET 4,(HL)	CB E6	2 / 3	- - - - - -
SET 4,(IX+d)	DD CB d E6	4 / 5	- - - - - -
SET 4,(IY+d)	FD CB d E6	4 / 5	- - - - - -
SET 4,A	CB E7	2 / 2	- - - - - -
SET 4,B	CB E0	2 / 2	- - - - - -
SET 4,C	CB E1	2 / 2	- - - - - -
SET 4,D	CB E2	2 / 2	- - - - - -
SET 4,E	CB E3	2 / 2	- - - - - -
SET 4,H	CB E4	2 / 2	- - - - - -
SET 4,L	CB E5	2 / 2	- - - - - -
SET 5,(HL)	CB EE	2 / 3	- - - - - -
SET 5,(IX+d)	DD CB d EE	4 / 5	- - - - - -
SET 5,(IY+d)	FD CB d EE	4 / 5	- - - - - -
SET 5,A	CB EF	2 / 2	- - - - - -
SET 5,B	CB E8	2 / 2	- - - - - -
SET 5,C	CB E9	2 / 2	- - - - - -
SET 5,D	CB EA	2 / 2	- - - - - -
SET 5,E	CB EB	2 / 2	- - - - - -
SET 5,H	CB EC	2 / 2	- - - - - -
SET 5,L	CB ED	2 / 2	- - - - - -
SET 6,(HL)	CB F6	2 / 3	- - - - - -
SET 6,(IX+d)	DD CB d F6	4 / 5	- - - - - -
SET 6,(IY+d)	FD CB d F6	4 / 5	- - - - - -
SET 6,A	CB F7	2 / 2	- - - - - -
SET 6,B	CB F0	2 / 2	- - - - - -
SET 6,C	CB F1	2 / 2	- - - - - -
SET 6,D	CB F2	2 / 2	- - - - - -
SET 6,E	CB F3	2 / 2	- - - - - -
SET 6,H	CB F4	2 / 2	- - - - - -
SET 6,L	CB F5	2 / 2	- - - - - -
SET 7,(HL)	CB FE	2 / 3	- - - - - -
SET 7,(IX+d)	DD CB d FE	4 / 5	- - - - - -
SET 7,(IY+d)	FD CB d FE	4 / 5	- - - - - -
SET 7,A	CB FF	2 / 2	- - - - - -
SET 7,B	CB F8	2 / 2	- - - - - -
SET 7,C	CB F9	2 / 2	- - - - - -
SET 7,D	CB FA	2 / 2	- - - - - -
SET 7,E	CB FB	2 / 2	- - - - - -
SET 7,H	CB FC	2 / 2	- - - - - -
SET 7,L	CB FD	2 / 2	- - - - - -
SLA (HL)	CB 26	2 / 5	s z h p n c
SLA (IX+d)	DD CB d 26	4 / 7	s z h p n c
SLA (IY+d)	FD CB d 26	4 / 7	s z h p n c
SLA A	CB 27	2 / 2	s z h p n c
SLA B	CB 20	2 / 2	s z h p n c
SLA C	CB 21	2 / 2	s z h p n c
SLA D	CB 22	2 / 2	s z h p n c
SLA E	CB 23	2 / 2	s z h p n c
SLA H	CB 24	2 / 2	s z h p n c
SLA L	CB 25	2 / 2	s z h p n c
SLP	ED 76	2 / 2	- - - - - -
SRA (HL)	CB 2E	2 / 5	s z h p n c
SRA (IX+d)	DD CB d 2E	4 / 7	s z h p n c
SRA (IY+d)	FD CB d 2E	4 / 7	s z h p n c
SRA A	CB 2F	2 / 2	s z h p n c
SRA B	CB 28	2 / 2	s z h p n c
SRA C	CB 29	2 / 2	s z h p n c
SRA D	CB 2A	2 / 2	s z h p n c
SRA E	CB 2B	2 / 2	s z h p n c
SRA H	CB 2C	2 / 2	s z h p n c
SRA L	CB 2D	2 / 2	s z h p n c
SRL (HL)	CB 3E	2 / 5	s z h p n c
SRL (IX+d)	DD CB d 3E	4 / 7	s z h p n c
SRL (IY+d)	FD CB d 3E	4 / 7	s z h p n c
SRL A	CB 3F	2 / 2	s z h p n c
SRL B	CB 38	2 / 2	s z h p n c
SRL C	CB 39	2 / 2	s z h p n c
SRL D	CB 3A	2 / 2	s z h p n c
SRL E	CB 3B	2 / 2	s z h p n c
SRL H	CB 3C	2 / 2	s z h p n c
SRL L	CB 3D	2 / 2	s z h p n c
STMIX	ED 7D	2 / 2	- - - - - -
SUB (HL)	96	2 / 2	s z h p n c
SUB (IX+d)	DD 96 d	2 / 4	s z h p n c
SUB (IY+d)	FD 96 d	2 / 4	s z h p n c
SUB A	97	1 / 1	s z h p n c
SUB B	90	1 / 1	s z h p n c
SUB C	91	1 / 1	s z h p n c
SUB D	92	1 / 1	s z h p n c
SUB E	93	1 / 1	s z h p n c
SUB H	94	1 / 1	s z h p n c
SUB IXH	DD AC	2 / 2	s z h p n c
SUB IXL	DD AD	2 / 2	s z h p n c
SUB IYH	FD AC	2 / 2	s z h p n c
SUB IYL	FD AD	2 / 2	s z h p n c
SUB L	95	1 / 1	- z v b
SUB n	D6 n	2 / 2	s z h p n c
TST A,(HL)	ED 34	2 / 3	s z h p n c
TST A,A	ED 3C	2 / 2	s z h p n c
TST A,B	ED 04	2 / 2	s z h p n c
TST A,C	ED 0C	2 / 2	s z h p n c
TST A,D	ED 14	2 / 2	s z h p n c
TST A,E	ED 1C	2 / 2	s z h p n c
TST A,H	ED 24	2 / 2	s z h p n c
TST A,L	ED 2C	2 / 2	s z h p n c
TST A,n	ED 64 n	3 / 3	s z h p n c
TSTIO n	ED 74 n	3 / 4	s z h p n c
XOR (HL)	AE	1 / 2	s z h p n c
XOR (IX+d)	DD AC d	3 / 4	s z h p n c
XOR (IY+d)	FD AC d	3 / 4	s z h p n c
XOR A	AF	1 / 1	s z h p n c
XOR B	A8	1 / 1	s z h p n c
XOR C	A9	1 / 1	s z h p n c
XOR D	AA	1 / 1	s z h p n c
XOR E	AB	1 / 1	s z h p n c
XOR H	AC	1 / 1	s z h p n c
XOR IXH	DD AC	2 / 2	s z h p n c
XOR IXL	DD AC	2 / 2	s z h p n c
XOR IYH	FD AD	2 / 2	s z h p n c
XOR IYL	FD AD	2 / 2	s z h p n c
XOR L	AD	1 / 1	s z h p n c
XOR n	EE n	2 / 2	s z h p n c
xxx.S	52 xxx	1 / 1	
xxx.L	49 xxx	1 / 1	
.IS	40	1 / 1	
.IS ADL	49	1 / 1	
.IL	52	1 / 1	
IL ADL	5B	1 / 1	
.SIS	40	1 / 1	
.LIL	5B	1 / 1	

SLL is removed

LD B,B C,C D,D E,E removed

6502

Mnemonic	Description	Example	Addressing Modes	Flags
ADC <ea>	Add <ea> and the carry flag to the Accumulator A.	ADC #61	Imm ; ZeroPg ; ZeroPg,X ; Abs ; Abs,X ; Abs,Y ; (ind) {65c02}; (Ind,X), (Ind),Y	N Z C - - V
AND <ea>	Logical AND of bits in 8 bit value <ea> with Accumulator	AND $12	Accum ; ZeroPg ; ZeroPg,X ; Abs ; Abs,X	N Z C - - -
ASL <ea>	Shift <ea> Left for Arithmetic.	ASL	Accum ; ZeroPg ; ZeroPg,X ; Abs ; Abs,X	- - - - - -
Bcc ofst	Branch to the 8 bit offset ofst IF condition cc is true.	BEQ TestLabel		- - - - - -
BIT <ea>	Test bits in Accumulator compared to <ea>	BIT $61	Imm {65c02} ; ZeroPg ; ZeroPg,X {65c02} ; Abs ; Abs,X {65c02}	N Z - - - V
BRK	Stop the CPU and execute an interrupt.	BRK		- - - I - -
CLC	Clear the Carry Flag. C flag will be set to Zero.	CLC		- - C - - -
CLD	Clear the Decimal Flag. (BCD off)	CLD		- - - - D -
CLI	Clear the Interrupt Flag. (Enable Interrupts)	CLI		- - - I - -
CLV	Clear the oVerflow Flag. V flag will be set to Zero.	CLV		- - - - - V
CMP <ea>	Compare the Accumulator to <ea>.	CMP #10	Imm ; ZeroPg ; ZeroPg,X ; Abs ; Abs,X ; Abs,Y ; (ind) {65c02} ; (Ind,X), (Ind),Y	N Z C - - -
CPX <ea>	Compare the X register to <ea>.	CPX #10	Imm ; ZeroPg ; Abs	N Z C - - -
CPY <ea>	Compare the Y register to <ea>.	CPY #10	Imm ; ZeroPg ; Abs	N Z C - -
DEC <ea>	Decrease the 8 bit value <ea> by one.	DEC $10	Accum {65c02}; ZeroPg ; ZeroPg,X ; Abs ; Abs,X	N Z - - - -
DEX	Decrease register X by one.	DEX		N Z - - - -
DEY	Decrease register Y by one.	DEY		N Z - - - -
EOR <ea>	Logical EOR (Exclusive OR) of bits in <ea> with A	EOR <ea>	Imp ; ZeroPg ; ZeroPg,X ; Abs ; Abs,X ; Abs,Y ; (ind) {65c02} ; (Ind,X), (Ind),Y	N Z - - - -
INC <ea>	Increase the 8 bit value <ea> by one.	INC $10	Accum {65c02}; ZeroPg ; ZeroPg,X ; Abs ; Abs,X	N Z - - - -
INX	Increase register X by one.	INX		N Z - - - -
INY	Increase register Y by one.	INY		N Z - - - -
JMP addr	Jump to the 16 bit address addr.	JMP $4000	Abs ; (Ind Abs,X) {65c02} ; (Ind)	- - - - - -
JSR addr	Jump to Subroutine at address addr.	JSR addr	Abs	- - - - - -
LDA <ea>	Load the 8 bit value from <ea> into the Accumulator.	LDA #100	Imm ; ZeroPg ; ZeroPg,X ; Abs ; Abs,X ; Abs,Y ; (ind) {65c02} ; (Ind,X), (Ind),Y	N Z - - - -
LDX <ea>	Load the 8 bit value from <ea> into the X register.	LDX #100	Imm ; ZeroPg ZeroPg,Y ; Abs ; Abs,Y	N Z - - - -
LDY <ea>	Load the 8 bit value from <ea> into the Y register.	LDY #100	Imm ; ZeroPg ZeroPg,X ; Abs ; Abs,X	N Z - - - -
LSR <ea>	Shift the bits of <ea> Right Logically.	LSR $1000	Accum ; ZeroPg ; ZeroPg,X ; Abs ; Abs,X	N Z C - - -
NOP	No Operation.	NOP		- - - - - -
ORA <ea>	Logical OR of bits in 8 bit value <ea> with Accumulator	ORA #61	Imm ; ZeroPg ; ZeroPg,X ; Abs ; Abs,X ; Abs,Y ; (ind) {65c02}; (Ind,X), (Ind),Y	N Z C - - V
PHA	Push a byte from register A onto the top of the stack.	PHA		- - - - - -
PHP	Push the flags (P) onto the stack.	PHP		- - - - - -
PLA	Pull a byte off the stack into register A.	PLA		- - - - - -
PLP	Pull a byte off the stack into register A.	PLP		N Z C I D V
ROL <ea>	Rotate bits of <ea> Left with the Carry.	ROL $40	Accum ; ZeroPg ; ZeroPg,X ; Abs ; Abs,X	N Z C - - -
ROR <ea>	Rotate bits of <ea> Right with the Carry.	ROR $40	Accum ; ZeroPg ; ZeroPg,X ; Abs ; Abs,X	N Z C - - -
RTI	Return from an interrupt.	RTI		N Z C I D V
RTS	Return from a subroutine.	RTS		- - - - - -
SBC <ea>	Subtract <ea> and the carry flag from the Accumulator	SBC #61	Imm ; ZeroPg ; ZeroPg,X ; Abs ; Abs,X ; Abs,Y ; (ind) {65c02}; (Ind,X), (Ind),Y	N Z C - - V
SEC	Set the carry flag to 1.	SEC		- - C - - -
SED	Set the Decimal Flag. (BCD on)	SED		- - - - D -
SEI	Set the Interrupt Flag.	SEI		- - - I - -
STA <ea>	Store the Accumulator into memory address <ea>.	STA $10	ZeroPg ; ZeroPg,X ; Abs ; Abs,X ; Abs,Y ; (ind) {65c02} ; (Ind,X), (Ind),Y	- - - - - -
STX <ea>	Store the X register into memory address <ea>.	STX $10	ZeroPg ; ZeroPg,Y ; Abs	- - - - - -
STY <ea>	Store the Y register into memory address <ea>.	STY $10	ZeroPg ; ZeroPg,X ; Abs	- - - - - -
TAX	Transfer the Accumulator into register X.	TAX		N Z - - - -
TAY	Transfer the Accumulator into register Y.	TAY		N Z - - - -
TSX	Transfer the Stack pointer into register X.	TSX		N Z - - - -
TXA	Transfer the X register into the Accumulator.	TXA		N Z - - - -
TXS	Transfer the X Register into the Stack pointer.	TXS		- - - - - -
TYA	Transfer the Y register into the Accumulator.	TYA		N Z - - - -

Mnemonic	Description	Example	Addressing Modes	Flags
BRA ofst	Branch always to the 8 bit offset ofst (without condition).	BRA TestLabel		- - - - - -
PHX	Push a byte from register X onto the top of the stack.	PHX		- - - - - -
PHY	Push a byte from register Y onto the top of the stack.	PHY		- - - - - -
PLY	Pull a byte off the stack into register Y.	PLY		- - - - - -
STZ <ea>	Clear the 8 bit value in memory address <ea>.	STZ $1000		- - - - - -

Mnemonic	Description	Condition
BCC label	Branch to label if Carry Clear	C=0
BCS label	Branch to label if Carry Set	C=1
BEQ label	Branch to label if Equal (Zero)	Z=1
BNE label	Branch to label if Not Equal (NonZero)	Z=0
BMI label	Branch to label if MInus	N=1
BPL label	Branch to label if PLus	N=0
BVC label	Branch to label if oVerflow Clear	V=0
BVS label	Branch to label if oVerflow Set	V=1

6502 – 65816 – 6280

		Implied	Relative	Accum	Immediate	Zero Page	Zero Pg,X	Zero PG,Y	Absolute	Absolute,X	Absolute,Y	Abs,X Indir	Indirect	(Indirect,X)	(Indirect),Y	C Z I D B V N	Absolute Long	Abs Indir Long	Direct Pg Indirect	Direct Pg Ind Lng	Abs Long,X	(Long Indirect),Y	Stack Relative	SR Indirect Indexed
					#nn	$nn	$nn,X	$nn,Y	$0100	$0100,X	$0100,Y	($nnnn,X)	($nnnn)	($nn,X)	($nn),Y		$100000	[$1000]	($nn)	[$nn]	$010000,X	[$nn],Y	$ss,S	($ss,S),Y
		no params	jr	works on A	&nn	(&00nn)	(&00nn+X)	(&00nn+Y)	(&0100)	(&0100+X)	(&0100+Y)	((&nnnn +X))	((&nnnn))	((&00nn+X))	((&00nn)+Y)		($100000)	(($1000))	($dpnn)	($dpdpnn)	(&010000+X)	((&dpdpnn)+Y)	($ss+S)	(($ss,S)+Y)
ADC	Add with Carry				$69 2 2	$65 2 3	$75 2 4		$6D 3 4	$7D 3 4/5	$79 3 4/5		*$72 3*	$61 2 6	$71 2 5/6	Ovf7 Z - - - +- 7	$6F 3 4		$72 2 5	$67 2 6	$7F 4 5	$77 2 6	$63 2 4	$73 2 7
AND	Logical AND				$29 2 2	$25 2 3	$35 2 4		$2D 3 4	$3D 3 4/5	$39 3 4/5		*$32 3*	$21 2 6	$31 2 5/6	- Z - - - - - 7	$2F 4 5		$32 2 5	$27 2 4	$3F 4 5	$37 2 6	$23 2 4	$33 2 7
ASL	Arithmetic Shift Left			$0A 1 2		$06 2 5	$16 2 6		$0E 3 6	$1E 3 7						7 Z - - - - - 7								
BCC	Branch if Carry Clear C=1 (Aka BLT)		$90 2 2-4													- - - - - - - -								
BCS	Branch if Carry Set C=0 (Aka BGE)		$B0 2 2-4													- - - - - - - -								
BEQ	Branch if Equal to Zero		$F0 2 2-4													- - - - - - - -								
BIT	Bit Test (set flags like AND)				$89 2	$24 2 3	$34 2		$2C 3 4	$3C 3						- Z - - - 6 7								
BMI	Branch if Minus (S = 1)		$30 2 2-4													- - - - - - - -								
BNE	Branch if Not Equal to Zero		$D0 2 2-4													- - - - - - - -								
BPL	Branch if Plus (S = 0)		$10 2 2-4													- - - - - - - -								
BRK	Break	$00 1 7														- - - - =1 - -								
BVC	Branch if Overflow Clear		$50 2 2-4													- - - - - - - -								
BVS	Branch if Overflow Set		$70 2 2-4													- - - - - - - -								
CLC	Clear Carry Flag															=0 - - - - - - -								
CLD	Clear Decimal Mode	$D8 1 2														- - - =0 - - -								
CLI	Clear Interrupt Mask (Enable Interrupts)	$58 1 2														- - =0 - - - -								
CLV	Clear Overflow Flag	$B8 1 2														- - - - - - =0 -								
CMP	Compare Accumulator to Memory				$C9 2 2	$C5 2 3	$D5 2 4		$CD 3 4	$DD 3 4/5	$D9 3 4/5		*$D2 3*	$C1 2 6	$D1 2 5/6	> = - - - - - 7	$CF 4 5		$D2 2 5	$C7 2 6	$DF 4 5	$D7 2 6	$C3 2 4	$D3 2 7
CPX	Compare with Index Register X				$E0 2 2	$E4 2 3			$EC 3 4							> = - - - - - 7								
CPY	Compare with Index Register Y				$C0 2 2	$C4 2 3			$CC 3 4							> = - - - - - 7								
DEC	Decrement (Aka DEA)			$3A		$C6 2 5	$D6 2 6		$CE 3 6	$DE 3 7						- Z - - - - - 7								
DEX	Decrement Index Register X	$CA 1 2														- Z - - - - - 7								
DEY	Decrement Index Register Y	$88 1 2														- Z - - - - - 7								
EOR	Logical Exclusive-OR (XOR)				$49 2 2	$45 2 3	$55 2 4		$4D 3 4	$5D 3 4/5	$59 3/4/5		*$52 3*	$41 2 6	$51 2 5/6	- Z - - - - - 7	$4F 4 5		$52 2 5	$47 2 6	$5F 4 5	$57 2 6	$43 2 4	$53 2 7
INC	Increment (Aka INA)			$1A		$E6 2 5	$F6 2 6		$EE 3 6	$FE 3 7						- Z - - - - - 7								
INX	Increment Index Register X	$E8 1 2														- Z - - - - - 7								
INY	Increment Index Register Y	$C8 1 2														- Z - - - - - 7								
JMP	Jump to New Location (or **JML** for long)								$4C 3 3			$7C 3	$6C 3 5			- - - - - - - -	$5C 4 4	$DC 3 6						
JSR	Jump to Subroutine (or **JSL** for long)								$20 3 6	$FC 3 8						- - - - - - - -	$22 4 8							
LDA	Load Accumulator				$A9 2 2	$A5 2 3	$B5 2 4		$AD 3 4	$BD 3 4/5	$B9 3 4/5		*$B2 3*	$A1 2 6	$B1 2 5	- Z - - - - - 7	$AF 4 5		$B2 2 5	$A7 2 6	$BF 4 5	$B7 2 6	$A3 2 4	#B3 2 7
LDX	Load Index Register X				$A2 2 2	$A6 2 3		$B6 2 4	$AE 3 4		$BE 3 4/5					- Z - - - - - 7								
LDY	Load Index Register Y				$A0 2 2	$A4 2 3	$B4 2 4		$AC 3 4	$BC 3 4/5						- Z - - - - - 7								
LSR	Logical Shift Right			$4A 1 2		$46 2 5	$56 2 6		$4E 3 6	$5E 3 7						- Z - - - - - 7								
NOP	No Operation	$EA 1 2														- - - - - - - -								
ORA	Logical (Inclusive) OR				$09 2 2	$05 2 3	$15 2 4		$0D 3 4	$1D 3 4/5	$19 3 4/5		*$12 3*	$01 2 6	$11 2 5/6	- Z - - - - - 7	$0F 4 5		$12 2 5	$07 2 6	$1F 4 5	$07 2 6	$03 2 4	$13 2 7
PHA	Push Accumulator onto Stack	$48 1 3														- - - - - - - -								
PHP	Push Processor Status	$08 1 3														- - - - - - - -								
PLA	Pull Accumulator from Stack	$68 1 4														- Z - - - - - 7								
PLP	Pull Processor Status	$28 1 4														S S S S S S S								
ROL	Rotate Left through Carry			$2A 1 2		$26 2 5	$36 2 6		$2E 3 6	$3E 3 7						old7 Z - - - - 7								
ROR	Rotate Right through Carry			$6A 1 2		$66 2 5	$76 2 6		$6E 3 6	$7E 3 7						old0 Z - - - - 7								
RTI	Return from Interrupt (RETI)	$40 1 6														S S S S S S S								
RTS	Return from Sub (RET) (or **RTL** for long)	$60 1 6 $6B 1 6														- - - - - - - -								
SBC	Subtract with Carry				$E9 2 2	$E5 2 3			$ED 3 4	$FD 3 4/5	$F9 3 4/5		$F2 3	$E1 2 6	$F1 2 5/6	Ovf7 Z - - - - 7	$EF 4 5		$F2 2 5	$E7 2 6	$FF 4 5	$F7 2 6	$E3 2 4	$F3 2 7
SEC	Set Carry (SCF)	$38 1 2														1 - - - - - - -								
SED	Set Decimal Flag	$F8 1 2														- - - 1 - - -								
SEI	Set Interrupt Mask (Disable Interrupts)	$78 1 2														- - 1 - - - -								
STA	Store Accumulator					$85 2 3	$95 2 4		$8D 3 4	$9D 3 5	$99 3 5		$92 3	$81 2 6	$91 2 6	- - - - - - - -	$8F 4 5		$92 2 5	$87 2 6	$9F 4 5	$97 2 6	$83 2 4	$93 $2 7
STX	Store Index Register X					$86 2 3		$96 2 4	$8E 3 4							- - - - - - - -								
STY	Store Index Register Y					$84 2 3	$94 2 4		$8C 3 4							- - - - - - - -								
TAX	Transfer Accumulator to Index Reg X	$AA 1 2														- Z - - - - 7								
TAY	Transfer Accumulator to Index Reg Y	$A8 1 2														- Z - - - - 7								
TSX	Transfer Stack Pointer to X	$BA 1 2														- Z - - - - 7								
TXA	Transfer Index Register X to Accumulator	$8A 1 2														- Z - - - - 7								
TXS	Transfer X to Stack Pointer	$9A 1 2														- - - - - - -								
TYA	Transfer Index Register Y to Accumulator	$98 1 2														- Z - - - - 7								
BRA	Branch Relative Always (JR) (**BRL** for long)		$80 2 $82 3 4													- - - - - - -								
COP	Coprocessor Enable	$02 2 7														- - I D - - -								
MVN	Block Move Next (LDIR) MVN M,N A bytes from MX->NY (Alters DBR)				$54 3 ?											- - - - - - -								
MVP	Block Move Previous (LDDR) MVP M,N A bytes from MX→NY (Alters DBR)				$44 3 ?											- - - - - - -								
PEA	Push Effective Absolute address							$F4 3 5								- - - - - - -								
PEI	Push Effective Indirect Address															- - - - - - -			$D4 2 6					
PER	Push effective PC Relative Indirect Addr		$62 3 6													- - - - - - -								
PHB	Push 8 bit Data Bank Reg (DBR)	$8B 1 3														- - - - - - -								
PHD	Push 16bit Direct Page Register	$0B 1 4														- - - - - - -								
PHK	Push 8 bit Program Bank Register (PBR)	$4B 1 3														- - - - - - -								
PHX	Push X	$DA 1														- - - - - - -								
PHY	Push Y	$5A 1														- - - - - - -								
PLB	Pull 8 bit Data Bank Reg (DBR)	$AB 1 4														- Z - - - - 7								
PLD	Pull 16bit Direct Page Register	$2B 1 5														- Z - - - - 7								
PLX	Pull X	$FA 1														- Z - - - - 7								
PLY	Pull Y	$7A 1														- Z - - - - 7								
REP	Reset Status Bits				$C2 2 3											? ? ? ? ? ? ? ?								
SEP	Set Status Bit				$E2 2 3											? ? ? ? ? ? ? ?								
STP	Stop processor until next RST	$DB 1														- - - - - - -								
STZ	Store Zero to address					$64 2	$74 2		$9C 3	$9E 3						- - - - - - -								
TCD	Transfer A to Direct page register (aka TAD)	$5B 1 2														- Z - - - - 7								
TDC	Transfer Direct Page register to A (aka TDA)	$7B 1 2														- Z - - - - 7								
TCS	Transfer Accumulator to SP (aka TAS)	$1B 1 2														- Z - - - - 7								
TRB	Test and Reset Bits with A					$14 2			$1C 3							- Z - - - - -								
TSB	Test and Set Bits with A					$04 2			$0C 3							- Z - - - - -								
TSC	Transfer SP to Accumulator (aka TSA)	$3B 1 2														- Z - - - - -								
TXY	Transfer X to Y	$9B 1 2														- Z - - - - 7								
TYX	Transfer Y to X	$BB 1 2														- Z - - - - 7								
WAI	Wait until any interrupt	$CB 1														- - - - - - -								
WDM	Reserved for future use!	$42 2														- - - - - - -								
XBA	Exchange A and B (aka SWA)	$EB 1 3														- Z - - - - 7								
XCE	Exchange Carry (C) and Emu bits (E)	$FB 1 2														E - - - M B - -								
BBR	Branch if bit n is Reset (also on some 65c02)					$0F-$7F 2										- - - - - - -								
BBS	Branch if bit n is Reset (also on some 65c02)					$8F-$FF 2										- - - D - - -								
BSR	Branch to subroutine (Call Relative)	$44 2 8																						
CLX	Clear X	$82 1 2																						
CLY	Clear Y	$C2 1 2																						
CSH	Change Speed High (7.16 MHz)	$D4 1 3																						
CSL	Change Speed Low (1.78 MHz)	$54 1 3																						
RMB	Reset Memory Bit n (also on some 65c02)					$07-$77										- - - - - - -								
SAX	Swap A and X	$22 1 3																						
SAY	Swap A and Y	$42 1 3																						
SET	Set T flag	$F4 1 2																						
SMB	Set Memory Bit n (also on some 65c02)					$87-$F7										- - - - - - -								
ST0	ST0 - Store (HuC6270) VDC No. 0				$03 2 5																			
ST1	ST1 - Store (HuC6270) VDC No. 1				$13 2 5																			
ST2	ST2 - Store (HuC6270) VDC No. 2				$23 2 5																			
SXY	Swap X and Y registers	$02 1 3																						
TAI	Transfer Alternate Increment				$F3 7 17+																			
TAM	Transfer Accumulator to MPR				$53 2 5																			
TIA	Transfer Increment Alternate				$E3 7 17+																			
TII	Transfer Increment Increment				$73 7 17+																			
TIN	Transfer Increment				$D3 7 17+																			
TMA	Transfer MPR to Accumulator				$43 2 4																			
TST	Test Bits at n2 with n1					$83 3 7	$A3 3 7	$93 4 8	$B3 4 8															

>=6502

65C02, 65816, 6280

65C02,6280 NOT 65816

6280

6280 + 65816

65816

16 bit in 65816 M=0 / X=0

68000

Mnemonic	Description	Example	Valid Lengths	Addressing Modes	Flags
ABCD Dm,Dn **ABCD -(Am),-(An)**	Adds two 8 bit Binary Coded Decimal numbers with eXtend	ABCD D1,D2	B		X n Z v C
ADD <ea>,Dn **ADD Dn,<ea>** **ADDA <ea>,An** **ADDI #,<ea>**	Adds two numbers together.	ADD D1,D2	B,W,L		X N Z V C
ADDQ #,<ea>	Adds a short immediate value # to <ea>.	ADDQ #1,A1	B,W,L		X N Z V C
AND <ea>,Dn **AND Dn,<ea>** **ANDI #,<ea>**	logically ANDs two numbers together.	AND D1,D2	B,W,L		X N Z V C
ANDI #,CCR	logically ANDs immediate value # with the CCR	ANDI #$F0,CCR	B		X N Z V C
ANDI ##,SR	is only available in Supervisor Mode.	ANDI #$0F,SR	W		X N Z V C
ASL Dm,Dn **ASL #<data>,Dn** **ASL <ea>**	Shift the bits Left for Arithmetic	ASL.W D1,D2	B,W,L		X N Z V C
ASR Dm,Dn **ASR #<data>,Dn** **ASR <ea>**	Shift the bits Right for Arithmetic	ASR.W D1,D2	B,W,L		X N Z V C
Bcc #	Branch to offset/Label # if the condition cc is true.	BCC TestLabel	B,W		- - - - -
BCHG Dn,<ea> **BCHG #,<ea>**	Test Bit Dn / # of destination <ea>, and flip bit in <ea>	BCHG #1,D1	B,L		- - Z - -
BCLR Dn,<ea> **BCLR #,<ea>**	Test Bit Dn / # of destination <ea>, and zero bit in <ea>	BCLR #1,D1	B,L		- - Z - -
BRA ofst	BRA TestLabel	BRA TestLabel	B,L		- - - - -
BSET Dn,<ea> **BSET #,<ea>**	Test Bit Dn / # of destination <ea>, and set bit in <ea>	BSET #1,D1	B,L		- - Z - -
BSR #	Branch to Subroutine at relative offset #.	BSR TestLabel	B,W		- - - - -
BTST Dn,<ea> **BTST #,<ea>**	Test Bit Dn or # of destination <ea>	BTST #1,D1	B,L		- - Z - -
CHK <ea>,Dn **CHK #,Dn**	Compare Dn to upper bound # Trap 6 if out of range	CHK #1000,D1	W, L {on 68020+}		- N z v c
CLR <ea>	Clear <ea> setting it to zero.	CLR.B $1000	B,W,L		- N Z V C
CMP <ea>,Dn **CMPA <ea>,An** **CMPI #,<ea>** **CMPM (Am)+,(An)+**	CMP compares <ea> to Dn.	CMPI.B #$FF,(A1)	B, W, L (W,L for CMPA)		- N Z V C
DBcc Dn,#	Decrease Dn, if Dn > -1 and branch if cc not true	DBRA D0,TestLabel	B,W		- - - - -
DIVS <ea>,Dn	Divide Signed numbers. Dn is divided by <ea>. Dn=Dn / <ea>.	DIVS #4,D1	L = L/w		- N Z V C
EOR Dn,<ea> **EORI #,<ea>**	Logical EOR (Exclusive OR) of bits in Dn or # with <ea>.	EOR #$20,D1	B,W,L		- N Z V C
EORI #,CCR	Logical EOR # with the CCR	EORI #$F0,CCR	B		X N Z V C
EORI ##,SR	is only available in Supervisor Mode.	EORI #$F0,SR			X N Z V C
EXG Dn,Dm **EXG An,Am**	Exchange the contents of registers Dn and Dm.	EXG D1,D2	L		- - - - -
EXT Dn	Sign extend register Dn, either extending a Byte to Word.	EXT.W D1	W,L		- N Z V C
ILLEGAL	execute "Illegal Instruction Vector" (Trap 4).	ILLEGAL			- - - - -
JMP #	Jump to absolute address #.	JMP TestLabel	L		- - - - -
JSR #	Jump to Subroutine at absolute address #.	JSR TestLabel	L		- - - - -
LEA <ea>,An	Load the effective address <ea> into An.	LEA (Label,PC),A1			- - - - -
LINK An,#	Creates a 'Tepmporary area' on the stack for work	LINK A1,#-4			- - - - -
LSL Dm,Dn **LSL #,Dn** **LSL <ea>**	Shift the bits in register Dn Left Logically by Dm or # bits.	LSL #1,D1			X N Z V C
LSR Dm,Dn **LSR #,Dn** **LSR <ea>**	Shift the bits in register Dn Right Logically by Dm or # bits.	LSR #1,D1	B, W, L		X N Z V C
MOVE <ea>,<ea2> **MOVEA <ea>,An**	Move the contents of source <ea> to the destination <ea2>.	MOVE #15,D1	B, W, L		- N Z V C
MOVE <ea>,CCR	moves a 16 bit value from <ea> to the CCR	MOVE D0,CCR	W		X N Z V C
MOVE SR,<ea> **MOVE <ea>,SR**	Move to or from the Status Register	MOVE SR,D0	W		X N Z V C
MOVE USP,An **MOVE An,USP**	Transfer the User Stack Pointer to or from address register An.	MOVE USP,A0	L		- - - - -
MOVEM <ea>,<Regs> **MOVEM <Regs>,<ea>**	The MOVEM command moves multiple registers	MOVEM.L (A1),D0/D3	B,W,L		- - - - -
MOVEP Dn,(#,An) **MOVEP (#,An),Dn**	Move 16 or 32 bits to a set of memory mapped byte data ports.	MOVEP.L D0,(4,A1)	W,L		- - - - -
MOVEQ #,Dn	adds short immediate # to the register Dn.	MOVEQ #1,D1	L		- - - - -
MULS <ea>,Dn	Multiply Signed numbers. Dn=Dn*<ea>.	MULS #4,D1	L=W*W		- N Z V C
MULU <ea>,Dn	Multiply Unsigned numbers. Dn=Dn*<ea>.	MULU #4,D1	L=W*W		- N Z V C
NBCD Dn	Negates BCD byte with eXtend. Dn. Dn=(0-Dn)-{X flag}	NBCD D1	B		X n Z v C
NEG <ea>	Negate <ea>	NEG D0	B, W, L		X N Z V C
NEGX <ea>	Negate <ea> with eXtend	NEGX <ea>	B, W, L		X N Z V C
NOP	No Operation.	NOP			- - - - - -
NOT <ea>	Invert/Flip all the bits of <ea>.	NOT.L D1	B, W, L		- N Z V C
OR <ea>,Dn **OR Dn,<ea>** **ORI #,<ea>**	logically ORs two numbers together.	OR D1,D2	B, W, L		- N Z V C
ORI #,CCR	logically ORs immediate value # with the CCR	ORI #$0F,CCR	B		X N Z V C
ORI ##,SR	logically ORs immediate value ## with the Status Register.	ORI #$0F,SR	W		X N Z V C
PEA <ea>,An	Push the effective address <ea> onto the stack.	PEA (Label,PC)	L		- - - - -
RESET	Sends an "RSTO" signal	RESET			- - - - -
ROL Dm,Dn **ROL #,Dn** **ROL <ea>**	Rotate bits in Dn to the Left by a number of bits	ROL.B #8,D1	B, W, L		- N Z V C
ROR Dm,Dn **ROR #,Dn** **ROR <ea>**	Rotate bits in Dn to the Right by a number of bits	ROR.B #8,D1	B, W, L		- N Z V C
ROXL Dm,Dn **ROXL #,Dn** **ROXL <ea>**	Rotate bits in Dn to the Left, with the eXtend bit	ROXL.B #8,D1	B, W, L		X N Z V C
ROXR Dm,Dn **ROXR #,Dn** **ROXR <ea>**	Rotate bits in Dn to the Right, with the eXtend bit	ROXR.B #8,D1	B, W, L		X N Z V C
RTE	Return from Exception.	RTE			X N Z V C
RTR	Return and Restore condition codes.	RTR			X N Z V C
RTS	Return from a Subroutine.	RTS			- - - - -
SBCD Dm,Dn **SBCD -(Am),-(An)**	Subtracts two 8 bit Binary Coded Decimal with eXtend carry	SBCD D1,D2	B		X n Z v C
Scc <ea>	Set <ea> to 255 or 0 according to condition cc.	SEQ.B TestLabel	B		- - - - -
STOP ##	Load the SR Status register with 16 bit immediate ## and halt				
SUB <ea>,Dn **SUB Dn,<ea>** **SUBA <ea>,An** **SUBI #,<ea>**	Subtracts two numbers.	SUBI.B #1,(A1)	B, W, L		X N Z V C
SUBQ #,<ea>	Subtracts a short immediate value # from <ea>.	SUBQ #1,A1	B, W, L		X N Z V C
SUBX Dm,Dn **SUBX -(Am),-(An)**	Subtracts with the eXtend bit.	SUBX D1,D2	B, W, L		X N Z V C
SWAP Dn	Swap the high and low words of register Dn.	SWAP D1	W		- N Z V C
TAS <ea>	Test and set <ea>.	TAS D1	B		- N Z V C
TRAP #	causes a jump to exception vector number #.	TRAP #1			- - - - -
TRAPV	If the oVerflow flag (V) is set, call overflow trap vector	TRAPV			- - - - -
TST <ea>	Set the flags according to <ea>.	TST.B D1	B, W, L		- N Z V C
UNLK An	Reverse the process performed by the LINK command.	UNLK An			- - - - -

cc	Description	Flags
CC	carry clear	C=0
CS	carry set	C=1
EQ	Equal	Z=1

GE	Greater than or equal	(N=1 & V=1) or (N=0 & V=0)
GT	Greater than	(N=1 & V=1 & Z=0) or (N=0 & V=0 & Z=0)
HI	Higher than	C=0 & Z=0
LE	Less than or equal	Z=1 or (N=1 & V=0) or (N=0 & V=1)
LS	Lower than or same	C=1 or Z=1
LT	Less than	(N=1 and V=0) or (N=0 and V=1)
MI	Minus	N=1
NE	Not equal	Z=0
PL	Plus	N=0
T	True	=0
F	False	=1
VC	Overflow clear	V=0
VS	Overflow set	V=1

6809

Cmd	Meaning	Inherent	Immediate	Direct	Exteded	Indx/Indir	Relative	H N Z V C
ABX	Add B to X	$3A (3/1)						– – – – –
ADCA	Add with Carry to A		$89 (2/2)	$99 (4/2)	$B9 (5/3)	$A9 (4+/2+)		* * * * *
ADCB	Add with Carry to B		$C9 (2/2)	$D9 (4/2)	$F9 (5/3)	$E9 (4+/2+)		* * * * *
ADDA	Add to A		$8B (2/2)	$9B (4/2)	$BB (5/3)	$AB (4+/2+)		* * * * *
ADDB	Add to B		$CB (2/2)	$DB (4/2)	$FB (5/3)	$EB (4+/2+)		* * * * *
ADDD	add to AB (16 bit)		$C3 (4/3)	$D3 (6/2)	$F3 (7/3)	$E3 (6+/2+)		– * * * *
ANDA	And with A		$84 (2/2)	$94 (4/2)	$B4 (5/3)	$A4 (4+/2+)		– * * 0 –
ANDB	And with B		$C4 (2/2)	$D4 (4/2)	$F4 (5/3)	$E4 (4+/2+)		– * * 0 –
ANDCC	And with ConditionCode		$1C (3/2)					– – – – 7
ASL	Arithmatic Shift Left			$08 (6/2)	$78 (7/3)	$68 (6+/2+)		8 * * * *
ASLA	Arithmatic Shift Left A	$48 (2/1)						8 * * * *
ASLB	Arithmatic Shift Left B	$58 (2/1)						8 * * * *
ASR	Arithmatic Shift Right			$07 (6/2)	$77 (7/3)	$67 (6+/2+)		8 * * – *
ASRA	Arithmatic Shift Right A	$47 (2/1)						8 * * – *
ASRB	Arithmatic Shift Right B	$46 (2/1)						8 * * – *
BCC	Branch if Carry Clear C=0						$24 (3/2)	– – – – –
BCS	Branch if Carry Set C=1						$25 (3/2)	– – – – –
BEQ	Branch if Equal Z=1						$27 (3/2)	– – – – –
BGE	Branch if Greater than or equal to zero						$2C (3/2)	– – – – –
BGT	Branch if Greater than Zero						$2E (3/2)	– – – – –
BHI	Branch if Higher Z+C=0						$22 (3/2)	– – – – –
BHS	Branch if Higher or Same C=0						$24 (3/2)	– – – – –
BITA	Bit Test A		$85 (2/2)	$95 (4/2)	$B5 (5/3)	$A5 (4+/2+)		8 * * 0 *
BITB	Bit Test B		$C5 (2/2)	$D5 (4/2)	$F5 (5/3)	$E5 (4+/2+)		8 * * 0 *
BLE	Branch if Less than or Equal to Zero						$2F (3/2)	– – – – –
BLO	Branch if Lower C=1						$25 (3/2)	– – – – –
BLS	Branch if Lower or Same C+Z=1						$23 (3/2)	– – – – –
BLT	Branch if Less Than Zero						$2D (3/2)	– – – – –
BMI	Branch if Minus N=1						$2B (3/2)	– – – – –
BNE	Branch if Not Equal to Zero Z=0						$26 (3/2)	– – – – –
BPL	Branch if Plus N=0						$2A (3/2)	– – – – –
BRA	Branch Always						$20 (3/2)	– – – – –
BRN	Branch Never						$21 (3/2)	– – – – –
BSR	Branch to Subroutine						$8D (3/2)	– – – – –
BVC	Branch if Overflow Clear V=0						$28 (3/2)	– – – – –
BVS	Branch if Overflow Set V=1						$29 (3/2)	– – – – –
CLR	Clear			$0F (6/2)	$7F (7/3)	$6F (6+/2+)		– 0 1 0 0
CLRA	Clear A	$4F (2/1)						– 0 1 0 0
CLRB	Clear B	$5F (2/1)						– 0 1 0 0
CMPA	Compare with A		$81 (2/2)	$91 (4/2)	$B1 (5/3)	$A1 (4+/2+)		8 * * * *
CMPB	Compare with B		$C1 (2/2)	$D1 (4/2)	$F1 (5/3)	$E1 (4+/2+)		8 * * * *
CMPD	Compare with AB		$10 83 (5/4)	$10 93 (7/3)	$10 B3 (8/4)	$10 A3 (7+/3+)		– * * * *
CMPS	Compare with S		$11 8C (5/4)	$11 9C (7/3)	$11 BC (8/4)	$11 AC (7+/3+)		– * * * *
CMPU	Compare with U		$11 83 (5/4)	$11 93 (7/3)	$11 B3 (8/4)	$11 A3 (7+/3+)		– * * * *
CMPX	Compare with X		$8C (4/3)	$9C (6/2)	$BC (7/3)	$AC (6+/2+)		– * * * *
CMPY	Compare with Y		$10 8C (5/4)	$10 9C (7/3)	$10 BC (8/4)	$10 AC (7+/3+)		– * * * *
COM	Complement			$03 (6/2)	$73 (7/3)	$63 (6/2)		– * * 0 1
COMA	Complement A	$43 (2/1)						– * * 0 1
COMB	Complement B	$53 (2/1)						– * * 0 1
CWAI	And with CC and Wait		$3C (20/2)					– – – – 7
DAA	Decimal Adjust after Addition	$19 (2/1)						– * * 0 *
DEC	Decrement			$0A (6/2)	$7A (7/3)	$6A (6+/2+)		– * * * –
DECA	Decrement A	$4A (2/1)						– * * * –
DECB	Decrement B	$5A (2/1)						– * * * –
EORA	Exclusive Or A (Xor)		$88 (2/2)	$98 (4/2)	$B8 (5/3)	$A8 (4+/2+)		– * * 0 –
EORB	Exclusive Or B (Xor)		$C8 (2/2)	$D8 (4/2)	$F8 (5/3)	$E8 (4+/2+)		– * * 0 –
EXG	Exchange Register Contents		$1E (8/2)					– – – – –
INC	Increment			$0C (6/2)	$7C (7/3)	$6C (6+/2+)		– * * * –
INCA	Increment A	$4C (2/1)						– * * * –
INCB	Increment B	$5C (2/1)						– * * * –
JMP	Jump			$0E (3/2)	$7E (4/3)		$6E (3+/2+)	– – – – –
JSR	Jump to Subroutine			$9D (7/2)	$BD (8/3)		$AD (7+/2+)	– – – – –
LBCC	Long Branch if Carry Clear C=0						$10 24 (5+/4)	– – – – –
LBCS	Long Branch if Carry Set C=1						$10 25 (5+/4)	– – – – –
LBEQ	Long Branch if Equal Z=1						$10 27 (5+/4)	– – – – –
LBGE	Long Branch if Greater than or equal to zero						$10 2C (5+/4)	– – – – –
LBGT	Long Branch if Greater than Zero						$10 2E (5+/4)	– – – – –
LBHI	Long Branch if Higher Z+C=0						$10 22 (5+/4)	– – – – –
LBHS	Long Branch if Higher or Same C=0						$10 24 (5+/4)	– – – – –
LBLE	Long Branch if Less than or Equal to Zero						$10 2F (5+/4)	– – – – –
LBLO	Long Branch if Lower C=1						$10 25 (5+/4)	– – – – –
LBLS	Long Branch if Lower or Same C+Z=1						$10 23 (5+/4)	– – – – –
LBLT	Long Branch if Less Than Zero						$10 2D (5+/4)	– – – – –
LBMI	Long Branch if Minus N=1						$10 2B (5+/4)	– – – – –
LBNE	Long Branch if Not Equal to Zero Z=0						$10 26 (5+/4)	– – – – –
LBPL	Long Branch if Plus N=0						$10 2A (5+/4)	– – – – –
LBRA	Long Branch Always						$16 (5/3)	– – – – –
LBRN	Long Branch Never						$10 21 (5/4)	– – – – –
LBSR	Long Branch to Subroutine						$17 (9/3)	– – – – –
LBVC	Long Branch if Overflow Clear V=0						$10 28 (5+/6)	– – – – –
LBVS	Long Branch if Overflow Set V=1						$10 29 (5+/6)	– – – – –
LDA	Load A		$86 (2/2)	$96 (4/2)	$B6 (5/3)	$A6 (4+/2+)		– * * 0 –
LDB	Load B		$C6 (2/2)	$D6 (4/2)	$F6 (5/3)	$E6 (4+/2+)		– * * 0 –
LDD	Load AB		$CC (3/3)	$DC (5/2)	$FC (6/3)	$EC (5+/2+)		– * * 0 –
LDS	Load S		$10 CE (4/4)	$10 DE (6/3)	$10 FE (7/4)	$10 EE (6+/3+)		– * * 0 –
LDU	Load U		$CE (3/3)	$DE (5/2)	$FE (6/3)	$EE (5+/2+)		– * * 0 –
LDX	Load X		$8E (3/3)	$9E (5/2)	$BE (6/3)	$AE (5+/2+)		– * * 0 –
LDY	Load Y		$10 8E (4/4)	$10 9E (6/3)	$10 BE (7/4)	$10 AE (6+/3+)		– * * 0 –
LEAS	Load Effective Address into S					$32 (4+/2+)		– – – – –
LEAU	Load Effective address into U					$33 (4+/2+)		– – – – –
LEAX	Load Effective Address into X					$30 (4+/2+)		– – * – –
LEAY	Load Effective Address into Y					$31 (4+/2+)		– – * – –
LSL	Logical Shift Left			$08 (6/2)	$78 (7/3)	$68 (6+/2+)		– * * * *
LSLA	Logical Shift Left A	$48 (2/1)						– * * * *
LSLB	Logical Shift Left B	$58 (2/1)						– * * * *
LSR	Logical Shift Right			$04 (6/2)	$74 (7/3)	$64 (6+/2+)		– 0 * – *
LSRA	Logical Shift Right A	$44 (2/1)						– 0 * – *
LSRB	Logical Shift Right B	$54 (2/1)						– 0 * – *
MUL	Multiply A*B – result in AB	$3D (11/1)						– – * – 9
NEG	Negate			$00 (6/2)	$70 (7/3)	$60 (6+/2+)		8 * * * *
NEGA	Negate A	$40 (2/1)						8 * * * *
NEGB	Negate B	$50 (2/1)						8 * * * *
NOP	No Operation	$12 2/1						– – – – –
ORA	Or A		$8A (2/2)	$9A (4/2)	$BA (5/3)	$AA (4+/2+)		– * * 0 –
ORB	Or B		$CA (2/2)	$DA (4/2)	$FA (5/3)	$EA (4+/2+)		– * * 0 –
ORCC	Or Condition Code		$1A (3/2)					– – – 7 –
PSHS	Push onto S stack (PC U Y X DP B A CC)		$34 (3/2)					– – – – –
PSHU	Push onto U stack (PC S Y X DP B A CC)		$36 (3/2)					– – – – –
PULS	Pull off S stack (PC U Y X DP B A CC)		$35 (3/2)					– – – – –
PULU	Pull off U stack (PC S Y X DP B A CC)		$37 (3/2)					– – – – –
ROL	Rotate Left through Carry			$09 (6/2)	$79 (7/3)	$69 (6+/2+)		– * * * *
ROLA	Rotate Left through Carry A	$49 (2/1)						– * * * *
ROLB	Rotate Left through Carry B	$59 (2/1)						– * * * *
ROR	Rotate Right through Carry			$06 (6/2)	$76 (7/3)	$66 (6+/2+)		– * * – *
RORA	Rotate Right through Carry A	$46 (2/1)						– * * – *
RORB	Rotate Right through Carry B	$56 (2/1)						– * * – *
RTI	Return from Interrupt	$3B (6/15)						– – – – 7
RTS	Return From Subroutine	$39 (5/1)						– – – – –
SBCA	Subtract with Carry from A		$82 (2/2)	$92 (4/2)	$B2 (5/3)	$A2 (4+/2+)		8 * * * *
SBCB	Subtract with Carry from B		$C2 (2/2)	$D2 (4/2)	$F2 (5/3)	$E2 (4+/2+)		8 * * * *
SEX	Sign Extend B into AB	$1D (2/1)						– * * 0 –
STA	Store A			$97 (4/2)	$B7 (5/3)	$A7 (4+/2+)		– * * 0 –
STB	Store B			$D7 (4/2)	$F7 (5/3)	$E7 (4+/2+)		– * * 0 –
STD	Store AB			$DD (5/2)	$FD (6/3)	$ED (5+/2+)		– * * 0 –
STS	Store S			$10 DF (6/3)	$10 FF (7/4)	$10 EF (6+/3+)		– * * 0 –
STU	Store U			$DF (5/2)	$FF (6/3)	$EF (5+/2+)		– * * 0 –
STX	Store X			$9F (5/2)	$BF (6/3)	$AF (5+/2+)		– * * 0 –
STY	Store Y			$10 9F (6/3)	$10 BF (7/4)	$10 AF (6+/3+)		– * * 0 –
SUBA	Subtract from A		$80 (2/2)	$90 (4/2)	$B0 (5/3)	$A0 (4+/2+)		8 * * * *
SUBB	Subtract from B		$C0 (2/2)	$D0 (4/2)	$F0 (5/3)	$E0 (4+/2+)		8 * * * *
SUBD	Subtract from AB		$83 (4/3)	$93 (6/2)	$B3 (7/3)	$A3 (6+/2+)		– * * * *
SWI	Software Interrupt	$3F (19/1)						– – – – –
SWI2	Software Interrupt 2	$10 3F (20/2)						– – – – –
SWI3	Software Interrupt 3	$11 3F (20/2)						– – – – –
SYNC	Syncronise to Ext Event (wait for interrupt)	$13 (2/1)						– – – – –
TFR	Transfer Register to Register (X,Y,U,S,A,B,D,PC,CC)		$1F (7/2)					– – – – –
TST	Test (Set flags)			$0D (6/2)	$7D (6/3)	$6D (6+/2+)		– * * 0 –
TSTA	Test A	$4D (2/1)						– * * 0 –
TSTB	Test B	$5D (2/1)						– * * 0 –

6309

Cmd	Meaning	Inherent	Immediate	Direct	Exteded	Indx/Indir	Relative	E F H I N Z V C
ADCD	Add Memory Word plus Carry with Accumulator D		$18 09 (4/4-5)	$10 99 (3/5-7)	$10 B9 (6-8)	$10 A9 (6-7/3)		– – – – * * * *
ADCR	Add Source Register plus Carry to Destination Register		$10 31(3/4)					– – – – * * * *
ADDE	Add Memory Byte to 8-Bit Accumulator E		$11 8B (3/3)	$11 9B (4-5/3)	$11 BB (3+/5+)	$11 AB (3+/5+)		– – * – * * * *
ADDF	Add Memory Byte to 8-Bit Accumulator F		$11 CB (3/3)	$11DB (5/4)	$11 FB (4/5-6)	$11 EB (3+/5+)		– – * – * * * *
ADDW	Add Memory Word to 16-Bit Accumulator W		$10 8B (4/4-5)	$10 9B (3/5-7)	$10 BB (4/6-8)	$10 AB (3+/6+)		– – – – * * * *
ADDR	Add Source Register to Destination Register		$10 30 (3 /4)					– – – – * * * *
AIM	Logical AND of Immediate Value with Memory Byte			$02 (3/6)	$72 (4/7)	$62 (3+/7+)		– – – – * * 0 –
ANDD	Logically AND Memory Word with Accumulator D		$10 84 (4/4-5)	$10 94 (3/5-7)	$10 B4 (4/6-8)	$10 A4 (3+/6+)		– – – – * * 0 –
ANDR	Logically AND Source Register with Destination Register		$10 34 (3/4)					– – – – * * 0 –
ASLD	Arithmetic Shift Left of Accumulator D	$10 84 (2/2-3)						– – * – * * – *
ASRD	Arithmetic Shift Right of Accumulator D	$10 3F (2/2-3)						– – – – * * – *
BAND	Logically AND Register Bit with Memory Bit			$11 30 (4/6-7)				– – – – – – – –
BEOR	Exclusive-OR Register Bit with Memory Bit			$11 34 (4/6-7)				– – – – – – – –
BIEOR	Exclusively-OR Register Bit with Inverted Memory Bit			$11 35 (4/6-7)				– – – – – – – –
BIOR	Logically OR Register Bit with Inverted Memory Bit			$11 33 (4/6-7)				– – – – – – – –
BITD	Bit Test Accumulator D with Memory Word Value		$10 85 (4/4-5)	$10 95 (3/5-7)	$10 B5 (4/6-8)	$10 A5 (3+/6+)		– – – – * * 0 –
BITMD	Bit Test the MD Register with an Immediate Value		$11 3C (3/4)					– – – – – * – –
BOR	Logically OR Memory Bit with Register Bit			$11 32 (4/6-7)				– – – – – – – –
CLRD	Load Zero into Accumulator		$10 4F (2/2-3)					– – – – 0 1 0 0
CLRE	Load Zero into Accumulator		$11 4F (2/2-3)					– – – – 0 1 0 0
CLRF	Load Zero into Accumulator		$11 5F (2/2-3)					– – – – 0 1 0 0
CLRW	Load Zero into Accumulator		$10 5F (2/2-3)					– – – – 0 1 0 0
CMPE	Compare Memory Byte from 8-Bit Accumulator		$11 81 (3/3)	$11 91 (3/4-5)	$11 B1 (3/5-6)	$11 A1 (3/4-5)		– – * – * * * *
CMPF	Compare Memory Byte from 8-Bit Accumulator		$11 C1 (3/3)	$11 D1 (3/4-5)	$11 F1 (3/5-6)	$11 E1 (3/4-5)		– – * – * * * *
CMPW	Compare Memory Word from 16-Bit Register		$10 81 (4/4-5)	$10 91 (3/5-7)	$10 B1 (4/6-8)	$10 A1 (3+/6+)		– – – – * * * *
CMPR	Compare Source Register from Destination Register		$10 37 (3/4)					– – – – * * * *
COMD	Complement Accumulator		$10 43 (2/2-3)					– – – – * * 0 1
COME	Complement Accumulator		$11 43 (2/2-3)					– – – – * * 0 1
COMF	Complement Accumulator		$11 53 (2/2-3)					– – – – * * 0 1
COMW	Complement Accumulator		$10 43 (2/2-3)					– – – – * * 0 1
DECD	Decrement Accumulator		$10 4A (2/2-3)					– – – – * * * –
DECE	Decrement Accumulator		$11 4A (2/2-3)					– – – – * * * –
DECF	Decrement Accumulator		$11 5A (2/2-3)					– – – – * * * –
DECW	Decrement Accumulator		$10 4A (2/2-3)					– – – – * * * –
DIVD	Signed Divide of Accumulator D by 8-bit value in Memory		$11 8D (3/25)	$11 9D (3/26-27)	$11 BD (4/27-28)	$11 AD (3+/27+)		– – – – * * * *
DIVQ	Signed Divide of Accumulator Q by 16-bit value in Memory		$11 8E (4/34)	$11 9E (3/25-36)	$11 BE (4/36-37)	$11 AE (3/36+)		– – – – * * * *
EIM	Exclusive-OR of Immediate Value with Memory Byte			$05 (3/6)	$65 (3+/7+)	$75 (4/7)		– – – – * * 0 –
EORD	Exclusively-OR Memory Word with Accumulator D		$10 88 (4/4-5)	$10 98 (3/5-7)	$10 B8 (4/6-8)			– – – – * * 0 –
EORR	Exclusively-OR Source Register with Destination Register		$10 36 (3/4)					– – – – * * 0 –
INCD	Increment Accumulator		$10 4C (2/2-3)					– – – – * * * –
INCE	Increment Accumulator		$11 4C (2/2-3)					– – – – * * * –
INCF	Increment Accumulator		$11 5C (2/2-3)					– – – – * * * –
INCW	Increment Accumulator		$10 5C (2/2-3)					– – – – * * * –
LDE	Load Data into 8-Bit Accumulator		$11 86 (3/3)	$11 96 (3/3)	$11 B6 (3/3)	$11 A6 (3/3)		– – – – * * 0 –
LDF	Load Data into 8-Bit Accumulator		$11 C6 (3/3)	$11 D6 (3/3)	$11 F6 (3/3)	$11 E6 (3/3)		– – – – * * 0 –
LDW	Load Data into 16-Bit Register		$10 86 (4/4)	$10 96 (3/5-6)	$10 B6 (4/6-7)	$10 A6 (3+/6+)		– – – – * * 0 –
LDBT	Load Memory Bit into Register Bit			$11 36 (4/6-7)				– – – – – – – –
LDMD	Load an Immediate Value into the MD Register		$11 3D (3/5)					– – – – – – – –
LDQ	Load 32-bit Data into Accumulator Q		$CD (5/5)	$10 DC (3/7-8)	$10 FC (4/8-9)	$10 EC (3+/8+)		– – – – * * 0 –
LSLD	Logical Shift Left of Accumulator D	$10 48 (2/2-3)						– – – – * * * *
LSRD	Logical Shift Right of 16-Bit Accumulator	$10 44 (2/2-3)						– – – – 0 * – *
LSRW	Logical Shift Right of 16-Bit Accumulator	$10 54 (2/2-3)						– – – – 0 * – *
MULD	Signed Multiply of Accumulator D and Memory Word		$11 8F (4/28)	$11 9F (3/29-30)	$11 BF (4/30-31)	$11 AF (3+/30+)		– – – – * * – –
NEGD	Negation (Twos-Complement) of Accumulator	$10 40 (2/2-3)						– – – – * * * *
OIM	Logical OR of Immediate Value with Memory Byte			$01 (3/6)	$71 (4/7)	$61 (3+/7+)		– – – – * * 0 –
ORD	Logically OR Accumulator D with Word from Memory		$10 8A (4/4-5)	$10 9A (3/5-7)	$10 BA (4/6-8)	$10 AA (3+/6+)		– – – – * * 0 –
ORR	Logically OR Source Register with Destination Register		$10 35 (3/4)					– – – – * * 0 –
PSHSW	Push Accumulator W onto the Hardware Stack	$10 38 (2/6)						– – – – – – – –
PSHUW	Push Accumulator W onto the User Stack	$10 3A (2/6)						– – – – – – – –
PULSW	Pull Accumulator W from the Hardware Stack	$10 39 (2/6)						– – – – – – – –
PULUW	Pull Accumulator W from the User Stack	$10 3B (2/6)						– – – – – – – –
ROLD	Rotate 16-Bit Accumulator Left through Carry	$10 49 (2/2-3)						– – – – * * * *
ROLW	Rotate 16-Bit Accumulator Left through Carry	$10 59 (2/2-3)						– – – – * * * *
RORD	Rotate 16-Bit Accumulator Right through Carry	$10 46 (2/2-3)						– – – – * * – *
RORW	Rotate 16-Bit Accumulator Right through Carry	$10 56 (2/2-3)						– – – – * * – *
SBCD	Subtract Memory Word and Carry from Accumulator D		$10 82 (4/4-5)	$10 92 (3/5-7)	$10 B2 (3+/6+)	$10 A2 (3+/6+)		– – – – * * * *
SBCR	Subtract Source Register and Carry from Destination Register		$10 33 (3/4)					– – – – * * * *
SEXW	Sign Extend a 16-bit Value in W to a 32-bit Value in Q	$14 (1/4)						– – – – * * – –
STE	Store 8-Bit Accumulator to Memory			$11 97 (3/4-5)	$11 B7 (4/5-6)	$11 A7 (3+/5+)		– – – – * * 0 –
STF	Store 8-Bit Accumulator to Memory			$11 D7 (3/4-5)	$11 F7 (4/5-6)	$11 E7 (3+/5+)		– – – – * * 0 –
STW	Store 16-Bit Register to Memory			$10 97 3/5-6)	$10 B7 (4/6-7)	$10 A7 (3+/6+)		– – – – * * 0 –
STBT	Store value of a Register Bit into Memory			$11 37 (4/7-8)				– – – – – – – –
STQ	Store Contents of Accumulator Q to Memory			$10 DD (3/7-8)	$10 FD(4/8-9)	$10 ED (3+/8+)		– – – – * * 0 –
SUBE	Subtract from value in 8-Bit Accumulator		$11 80 (3/3)	$11 90 (3/4-5)	$11 B0 (4/5-6)	$11 A0 (4/5-6)		– – * – * * * *
SUBF	Subtract from value in 8-Bit Accumulator		$11 C0 (3/3)	$11 D0 (3/4-5)	$11 F0 (4/5-6)	$11 E0 (4/5-6)		– – * – * * * *
SUBW	Subtract from value in 16-Bit Accumulator		$10 80 (4/4-5)	$10 90 (3/5-7)	$10 B0 (4/6-8)	$10 A0 (3+/6+)		– – – – * * * *
SUBR	Subtract Source Register from Destination Register		$10 32 (3/4)					– – – – * * * *
TFM ++	Transfer Memory		$11 38 (3/9+)					– – – – – – – –
TFM --	Transfer Memory		$11 39 (3/9+)					– – – – – – – –
TFM +x	Transfer Memory		$11 3A (3/9+)					– – – – – – – –
TFM x+	Transfer Memory		$11 3B (3/9+)					– – – – – – – –
TIM	Bit Test Immediate Value with Memory Byte			$0B (3/6)	$7B (4/7)	$6B (3+/7+)		– – – – * * 0 –
TSTD	Test Value in Accumulator	$10 4D (2/2-3)						– – – – * * 0 –
TSTE	Test Value in Accumulator	$11 4D (2/2-3)						– – – – * * 0 –
TSTF	Test Value in Accumulator	$11 5D (2/2-3)						– – – – * * 0 –
TSTW	Test Value in Accumulator	$10 5D (2/2-3)						– – – – * * 0 –

8086

Mnemonic	Description	Example	Valid Regs	Flags affected
AAA	ASCII Adjust for Addition. Treats AL as an unpacked binary coded decimal number	AAA		o s z A p C
AAD	ASCII Adjust for Division. AL=AL+(AH*10), AH=0.	AAD		o S Z a P c
AAM	ASCII Adjust for Multiplication. We can use the normal MUL command then use AAM	AAM		o S Z a P c
AAS	ASCII Adjust for Subtraction. This treats AL as an unpacked binary coded decimal number	AAS		o s z A p C
ADC dest,src	Add src and the carry flag to dest.	ADC CX,1000h		O S Z A P C
ADD dest,src	Add src to dest.	ADD CX,1000h		O S Z A P C
AND dest,src	Logical AND of bits in dest with Accumulator scr.	AND AX,1100h		O S Z A P C
CALL dest	Call Subroutine at address dest.	CALL 1000h		- - - - - -
CBW	Convert the 8 bit byte in AL into a 16 bit word in AX.	CBW		- - - - - -
CLC	Clear the Carry Flag. C flag will be set to Zero.	CLC		- - - - - C
CLD	Clear the Direction Flag. D flag will be set to Zero. This is used for 'String functions'.	CLD		D - - - - - -
CLI	Clear the Interrupt enable flag. I flag will be set to 0. This disables maskable interrupts.	CLI		I - - - - - -
CMC	Complement the Carry flag. If C=1 it will now be 0. If it was 0 it will now be 1.	CMC		- - - - - C
CMP dest,src	Compare the Byte or Word dest to src. This sets the flags the same as "SUB dest,src" would.	CMP AL,32		O S Z A P C
CMPSBCMPSW	Compare DS:SI to ES:DI. This command can work in bytes or words. Sets flags like CMP	REPZ CMPSB		O S Z A P C
CWD	Convert the 16 bit word in AX into a 32 bit doubleword in DX.AX. This 'Sign Extends' AX	CWD		- - - - - -
DAA	Decimal Adjust for Addition. This treats AL as a packed binary coded decimal number.	DAA		O S Z A P C
DAS	Decimal Adjust for Subtraction. This treats AL as a packed binary coded decimal number.	DAS		O S Z A P C
DEC Dest	Divide Unsigned number AX or DX.AX by src.	DEC AL		O S Z A P -
DIV src	Divide Unsigned number AX or DX.AX by src. AL=AX/src (8 bit) or AX=DX.AX/src (16 bit)	DIV CX		o s z a p c
ESC #,src	This command is for working with multiple processors - it's not something you will need.	ESC 1,AH		- - - - - -
HLT	Stop the CPU until an interrupt occurs	HLT		- - - - - -
IDIV src	Divide Signed number AX or DX.AX by src. AL=AX / src (8 bit) or AX=DX.AX / src (16 bit)	IDIV CX		o s z a p c
IMUL src	Multiply Signed number AX or DX.AX by src. AX=AL*src (8 bit) or DX.AX=AX*src (16 bit)	IMUL CX		O s z a p C
IN dest,port	Read in an 8 bit byte or 16 bit word into dest (either AX, AL or AH). Use DX for 16 bit port num	IN AX,F0h		- - - - - -
INC Dest	Increase Dest by one. This is faster than using ADD with a value of 1.	INC AL		O S Z A P -
INT #	Causes software interrupt #. The flags are pushed onto the stack before call	INT 33h		- - - - - -
INTO	INTO will cause Interrupt 4 if the Overflow flag (O) is set, otherwise it will have no effect.	INTO		- - - - - -
IRET	Restore the flags from the stack and return from an Interrupt.	IRET		O S Z A P C
Jcc addr	Jump to 8 bit offset addr if condition cc is true.	JO ErrorHandle		- - - - - -
JCXZ addr	Jump to 8 bit offset addr if CX=0.	JCXZ NoLoop		- - - - - -
JMP addr	Jump to address addr.	JMP BX		- - - - - -
LAHF	Load AH from the Flags. This only transfers the main flags: SZ-A-P-C	LAHF		- - - - - -
LDS reg,addr	Load a full 32 bit pointer into DS segment register and register reg.	LDS BX,TestPointer	AX, BX, CX, DX, SI, DI	- - - - - -
LEA reg,src	Load the effective address src into reg.	LEA CX,[BX+DI]	AX, BX, CX, DX,SI, DI,	- - - - - -
LES reg,addr	Load a full 32 bit pointer into ES segment register and register reg.	LES AX,MyLabel	AX,BX,CX,DX,SI,DI	- - - - - -
LOCK	Enable the LOCK signal. This is for multiprocessor systems.	LOCK		- - - - - -
LODSBLODSW	Load from DS:SI into AX or AL. This command can work in bytes or words.	LODSB		- - - - - -
LOOP addr	Decrease CX and jump to label addr if CX is not zero.	LOOP LoopLabel		- - - - - -
LOOPNZ addr **LOOPNE addr**	Decrease CX and jump to label addr if CX is not zero and the Zero flag is not set.	LOOPNZ LoopLabel		- - - - - -
LOOPZ addr **LOOPE addr**	Decrease CX and jump to label addr if CX is not zero and the Zero flag is set.	LOOPZ LoopLabel		- - - - - -
MOV dest,src	Move a value from source src to destination dest.	MOV AX,BX		- - - - - -
MOVSB **MOVSW**	Move a byte or word from DS:SI to ES:DI. This command can be combined with repeat command REP, to repeat CX times.	REPZ MOVSB		- - - - - -
MUL src	Multiply unsigned number AX or DX.AX by src.AX=AL*src (8 bit) or DX.AX=AX*src (16 bit)	MUL CX		O s z a p C
NEG dest	Negate dest (Twos Complement of the number).	NEG AL		- - - - - -
NOP	No Operation. This command has no effect on any registers or memory.	NOP		- - - - - -
NOT dest	Invert/Flip all the bits of dest.	NOT dest		- - - - - -
OR dest,src	Logically ORs the src and dest parameter together.	OR AX,BX		O S Z a P C
OUT port,src	Send an 8 bit byte or 16 bit word from src (either AX or AL) to hardware port number port.	OUT 100,AL		- - - - - -
POP reg	Pop a pair of bytes off the stack into 16 bit register reg.	POP ES	AX, BX, CX, DX, SI, DI,	- - - - - -
POPF	Pop a pair of bytes off the stack into the 16 bit Flags register.	POPF		O D I T S Z A P C
PUSH reg	Push a pair of bytes from 16 bit register reg onto the top of the stack.	PUSH AX		- - - - - -
PUSHF	Push a pair of bytes off the stack into the 16 bit Flags register.	PUSHF		- - - - - -
RCL dest,count	Rotate bits in Destination dest to the Left by count bits, with the carry flag acting as an extra bit.	RCL AX,1		O - - - - C
RCR dest,count	Rotate bits in Destination dest to the Right by count bits, with the carry flag acting as an extra bit	RCR AX,1		O - - - - C
REP stringop	Repeat string operation stringop while CX>0. Decrease CX after each iteration	REP LODSW		- - - - - -
REPE stringop **REPZ stringop**	Repeat string operation stringop while the Z flag is set and CX>0. Decrease CX each time	REPZ CMPSB		- - - - - -
REPNE stringopI	Repeat string operation stringop while the Z flag is not set and CX>0. Decrease CX each time	REPNZ CMPSB		- - - - - -
RET	Return from a subroutine.	RET		- - - - - -
ROL dest,count	Rotate bits in Destination dest to the Left by count bits	ROL AX,1		O - - - - C
ROR dest,count	Rotate bits in Destination dest to the Right by count bits	ROR AL,1		O - - - - C
SAHF	Store AH to the Flags. This only transfers the main flags: SZ-A-P-C .	SAHF		- S Z A P C
SAL dest,count	Shift the bits for Arithmetic in Destination dest to the Left by count bits.	SAL AX,1		O - - - - C
SAR dest,count	Shift the bits for Arithmetic in Destination dest to the Right by count bits.	SAR AX,1		O - - - - C
SBB dest,src	Subtract src and the Borrow (carry flag) from dest.	SBB AL,BL		O S Z A P C
SCASBSCASW	Scan ES:DI and compare to AX or AL. This command can work in bytes or words. (Like CMP)	REPZ SCASB		O S Z A P C
SHL dest,count	Shift the bits logically Left in destination dest by count bits.	SHL AX,1		O - - - - C
SHR dest,count	Shift the bits logically Right in destination dest by count bits.	SHR AX,1		O - - - - C
STC	Set the Carry Flag. C flag will be set to 1.	STC		- - - - - C
STD	Set the Direction Flag. D flag will be set to 1. This is used for 'String functions'.	STD		D - - - - - -
STI	Set the Interrupt enable flag. I flag will be set to 1. This enables maskable interrupts.	STI		I - - - - - -
STOSBSTOSW	Store AX or AL to ES:DI. This command can work in bytes or words.	REP STOSB		- - - - - -
SUB dest,src	Subtract src from dest.	SUB AX,BX		O S Z A P C
TEST dest,src	Test dest, setting the flags in the same way a logical "AND src" would. Dest unchanged	TEST BX,64h		O S Z A P C
WAIT	Wait until the busy pin of the CPU is inactive.	WAIT		O S Z A P C
XCHG reg1,reg2	Exchange the contents of registers reg1 and reg2.	XCHG BH,AL		- - - - -
XLAT	Translate AL using lookup table DS:BX. AL is read from memory address [DS:BX+AL].	XLAT		- - - - - -

Command	Details	Flags
JA / JNBE	Above / Not Below or Equal (For Unsigned Numbers)	C=0 AND Z=0
JBE / JNA	Below or Equal / Not Above (For Unsigned Numbers)	C=1 OR Z=1
JC JB / JNAE	Carry Below / Not Above or Equal (For Unsigned Numbers)	C=1
JE / JZ	Equal / zero	Z=1
JG / JNLE	Greater / Not Less than or Equal (For Signed Numbers)	((S XOR O) OR Z)=0

JGE / JNL	Greater or Equal / Not Less (For Signed Numbers)	(S XOR O)=0	
JLE / JNG	Less than or Equal / Not Greater (For Signed Numbers)	((S XOR O) OR Z)=1	
JL / JNGE	Less / Not Greater or Equal (For Signed Numbers)	(S XOR O)=1	
JNC JAE / JNB	No CarryAbove or Equal / Not Below (For Unsigned Numbers)	C=0	
JNE / JNZ	Not Equal / Not zero	Z=0	
JNO	Not Overflow	O=0	
JNP / JPO	Not Parity / Parity Odd	P=0	
JNS	Not signed (not negative)	S=0	
JO	Overflow	O=1	
JP / JPE	Parity / Parity Equal (bits 0-7 only)	P=1	
JS	Signed (is positive)	S=1	

PDP-11

				Opcode (Octal)						
Mnemonic	Function	Notes	N Z V C	F	E D C	B A 9	8 7 6	5 4 3	2 1 0	
HALT	Stop		- - - -	0	0	0	0	0	0	
WAIT	Stop until interrupt		- - - -	0	0	0	0	0	1	
RESET	Reset all IO devices		- - - -	0	0	0	0	0	5	
NOP	NoOp		- - - -	0	0	0	2	4	0	
CLR{B} dest	Set to zero		0 1 0 0	0	0	5	0	*D*	*D*	
INC{B} dest	Add 1		* * * -	0	0	5	2	*D*	*D*	
DEC{B} dest	Sub 1		* * * -	0	0	5	3	*D*	*D*	
ADC{B} dest	Add with carry		* * * *	0	0	5	5	*D*	*D*	
SBC{B} dest	Subtract with Carry		* * * *	0	0	5	6	*D*	*D*	
TST{B} dest	Set Condition Codes		* * 0 0	0	0	5	7	*D*	*D*	
NEG{B} dest	Negate		* * * *	0	0	5	4	*D*	*D*	
COM{B} dest	Ones compliment		* * 0 1	0	0	5	1	*D*	*D*	
ROR{B} dest	Rotate Right (through Carry)		* * * *	0	0	6	0	*D*	*D*	
ROL{B} dest	Rotate Left (through Carry)		* * * *	0	0	6	1	*D*	*D*	
ASR{B} dest	Arithmatic shift Right		* * * *	0	0	6	2	*D*	*D*	
ASL{B} dest	Arithmatic shift Left		* * * *	0	0	6	3	*D*	*D*	
SWAB	Swap Bytes in a word		* * * 0	0	0	0	3	*D*	*D*	
SXT	Sign Extend		- * 0 -	0	0	6	7	*D*	*D*	
MUL s,d	Multiply (if even registers 32 bit (r0+r1) else 16 (r1)		* * 0 *	0	7	0	*R*	*S*	*S*	mul r0,r2 ;R2.R3=R0*R2
DIV src,dest	Divide (dest reg must be even - r1=result r0=remaind		* * * *	0	7	1	*R*	*S*	*S*	
ASH n,reg	Arithmatic shift (by n bits) (n Positive=Right Neg=Left		* * * *	0	7	2	*R*	*S*	*S*	
ASHC n,reg	Arithmatic shift combined (32 bit pair)		* * * *	0	7	3	*R*	*S*	*S*	
XOR src,dest	Flip bits of dest with src		* * 0 -	0	7	4	*R*	*S*	*S*	div r0,r2 ;R2=R2.R3/R0 Rmdr in R3
MOV{B} src,dest	Move src to dest			B	1	*S*	*S*	*D*	*D*	
ADD src,dest	Add src to dest			0	6	*S*	*S*	*D*	*D*	
SUB src,dest	Subtract s from d			1	6	*S*	*S*	*D*	*D*	
CMP{B} src,dest	Compare (set flags like src-dest)			B	2	*S*	*S*	*D*	*D*	
BIS{B} src,dest	Bit Set (OR)			B	5	*S*	*S*	*D*	*D*	
BIC{B} src,dest	Bit Clear (for AND use with COM/ ^C to flip bits)			B	4	*S*	*S*	*D*	*D*	
BIT{B} src,dest	Bit Test (like AND but doesn't alter dest)			B	3	*S*	*S*	*D*	*D*	
BR ofst	Branch Always			0	0	0	1 *B B*	*B*	*B*	
BNEofst	Branch Not Equal	Z=0		0	0	1	0 *B B*	*B*	*B*	
BEQ ofst	Branch Equal	Z=1		0	0	1	1 *B B*	*B*	*B*	
BPL ofst	Branch if plus	N=0		1	0	0	0 *B B*	*B*	*B*	
BMI ofst	Branch if minus	N=1		1	0	0	1 *B B*	*B*	*B*	
BVC ofst	Branch if Overflow Clear	V=0		1	0	2	0 *B B*	*B*	*B*	
BVS ofst	Branch if Overflow Set	V=1		1	0	2	1 *B B*	*B*	*B*	
BHIS ofst	Branch if higher or same	C=0		1	0	3	0 *B B*	*B*	*B*	
BCC ofst	Branch if carry clear	C=0		1	0	3	0 *B B*	*B*	*B*	
BLO ofst	Branch if lower	C=1		1	0	3	1 *B B*	*B*	*B*	
BCS ofst	Branch if carry set	C=1		1	0	3	1 *B B*	*B*	*B*	
BGE ofst	Branch if greater than or equal to	N xor V=0		0	0	2	0 *B B*	*B*	*B*	
BLT ofst	Branch if less than	N xor V=1		0	0	2	1 *B B*	*B*	*B*	
BGT ofst	Branch if greater than	not (N xor V)=0		0	0	3	0 *B B*	*B*	*B*	
BLE ofst	Branch on less than or equal to	not (N xor V)=1		0	0	3	1 *B B*	*B*	*B*	
BHI ofst	Branch on higher than	C not Z =0		1	0	1	0 *B B*	*B*	*B*	
BLOS ofst	Branch on lower than or same as	C not Z = 1		1	0	1	1 *B B*	*B*	*B*	
JMP dest	Jump			0	0	0	1	*A*	*A*	
SOB	Subtract 1 and branch			0	7	7	*R*	*N*	*N*	
JSR reg,Label	Jump to subroutine, setting reg to return address			0	0	4	*R*	*A*	*A*	
RTS reg	Return from subroutine to address reg, and pop reg from the stack			0	0	0	2	0	*R*	
RTI	Return from interrupt/trap			0	0	0	0	0	2	
TRAP	Trap	T>=400		1	0	4	*T*	*T*	*T*	
BPT	Breakpoint trap			0	0	0	0	0	3	
IOT	I/O Trap			0	0	0	0	0	4	
EMT	Emulator Trap	T<=400		1	0	4	*T*	*T*	*T*	
RTT	Return from trace trap			0	0	0	0	0	6	
SPL	Set priority level			0	0	0	2	3	*N*	
-	Clear Multiple			0	0	0	2	1 0 *N*	*Z V C*	
CLC	Clear Carry flag	C=0		0	0	0	2	4	1	
CLV	Clear Overflow flag	V=0		0	0	0	2	4	2	
CLZ	Clear Zero flag	Z=0		0	0	0	2	4	4	
CLN	Clear Negative flag	N=0		0	0	0	2	5	0	
CCC	Clear Condition codes	All=0		0	0	0	2	5	7	
-	Set Multiple			0	0	0	2	1 1 *N*	*Z V C*	
SEC	Set Carry flag	C=1		0	0	0	2	6	1	
SEV	Set Overflow flag	V=1		0	0	0	2	6	2	
SEZ	Set Zero flag	Z=1		0	0	0	2	6	4	
SEN	Set Negative flag	N=1		0	0	0	2	7	0	
SCC	Set condition codes	All=1		0	0	0	2	7	7	
				1	4 2 1	4 2 1	4 2 1	4 2 1	4 2 1	
.ASCII	Ascii									
.ASCIZ	Ascii followed by zero byte									
ALIGN	Base address									
ORG	Base address									
.BYTE	Byte data									
.WORD	Word data									
.BLKW n	output n zero words									
.BLKB n	output n zero bytes									
.END	end of source code									
.INCLUDE	include another file									
CALL addr	Call subroutine – same as JSR PC,n									
RETURN	Return from subroutine -same as RTS PC									
MFPS r	Move from Processor status to register r									
MTPS r	move to Processor Status from reg r									

{B}=Byte (0=word / 1 = Byte)

TMS9900

Opcode	Meaning	Bytes	Fmt	L A = C V P X	Details	Example
A S,D	Add	A000	1	* * * * * - -		A @>100,R2
AB S,D	Add Bytes	B000	1	* * * * * * -		
C S,D	Compare	8000	1	* * * - - - -		
CB S,D	Compare Bytes	9000	1	* * * - - * -		CB R1,R2
S S,D	Subtract	6000	1	* * * * * - -		
SB S,D	Subtract Bytes	7000	1	* * * * * * -		
SOC	Set ones Corresponding (OR)	E000	1	* * * - - - -		
SOCB	Set ones Corresponding Bytes (OR)	F000	1	* * * - - * -		
SZC	Set Zeros Corresponding (Reverse AND)	4000	1	* * * - - - -		
SZCB	Set Zeros Corresponding Bytes (Reverse AND)	5000	1	* * * - - * -		
MOV S,D	Move	C000	1	* * * - - - -		
MOVB S,D	Move Bytes	D000	1	* * * - - * -		
COC S,D	Compare Ones Corresponding	2000	3	- - * - - - -	ones in S also in D?	COC R1O,RIl
CZC S,D	Compare Zeros Corresponding	2400	3	- - * - - - -		
XOR S,D	Flip Bits	2800	3	* * * - - - -		
MPY S,D	Multiply s*d – result in d,d+1	3800	9	- - - - - - -		MPY R2,R3
DIV Ss,D	Divide d,d+1 by s, result in d,d+1	3C00	9	- - - - * - -		DIV @>FEOO,R5
XOP A,n	Extend Operation	2800	9	2 2 2 2 2 2 2	Load new settings from address at vector A	XOP @>FFOO,4
B R	Branch to register R / @addr	0440	6	- - - - - - -	R->PC	B *R2
BL A	Branch and Link address A	0680	6	- - - - - - -	PC→WR11, SA→PC	
BLWP	Branch and Load Workspace Pointer	0400	6	- - - - - - -	(A)→WP (A+2)→PC ST→R15, PC→R14, WP→R13 (addr is 2 pntrs)	
CLR D	Clear Operand	04C0	6	- - - - - - -		
SETO	Set To Ones	0700	6	- - - - - - -		
INV D	Invert	0540	6	* * * - - - -		
NEG D	Negative	0500	6	* * * * * - -		
ABS D	Absolute Value	0740	6	* * * * * - -		
SWPB D	Swap Bytes	06C0	6	- - - - - - -		
INC D	Increment	0580	6	* * * * * - -		
INCT D	Increment by 2	05C0	6	* * * * * - -		
DEC D	Decrement	0600	6	* * * * * - -		
DECT D	Decrement by 2	0640	6	* * * * * - -		
X D	Execute	0480	6	2 2 2 2 2 2 2	.	
LDCR S,B	Load Communication Register…	3000	4	* * * - - 1 -	Transfer B bits from S	
STCR S,B	Store Communication Register	3400	4	* * * - - 1 -	Transfer B bits from S	
SBO n	Set CRU Bit to 1	1D00	x	- - - - - - -		SBO 4
SBZ n	Set CRU Bit to 0	1E00	x	- - - - - - -		
TB n	Test CRU Bit	1F00	x	- - * - - - -		
JEQ n	Jump Equal	1300	2	- - - - - - -	Jump to offset n	JEQ $+4
JGT	Jump Greater Than (Signed)	1500	2	- - - - - - -		
JH	Jump High	1B00	2	- - - - - - -		
JHE	Jump Higher or Equal	1400	2	- - - - - - -		
JL	Jump Lower	1A00	2	- - - - - - -		
JLE	Jump Lower or Equal	1200	2	- - - - - - -		
JLT	Jump Less Than (Signed)	1100	2	- - - - - - -		
JMP	Jump	1000	2	- - - - - - -		JMP $
JNC	Jump No Carry	1800	2	- - - - - - -		
JNE	Jump Not Equals	1600	2	- - - - - - -		
JNO	Jump No Overflow	1900	2	- - - - - - -		
JOC	Jump On Carry	1800	2	- - - - - - -		
JOP	Jump Odd Parity	1C00	2	- - - - - - -		
SLA D,B	Shift Left Arithmatic	0A00	5	* * * * * - -	Shift D by B bits (0=use R0)	SLA RI,O
SRA D,B	Shift Right Arithmatic	0800	5	* * * * - - -	Shift D by B bits (0=use R0)	SRA RI,2
SRC D,B	Shift Right Circular	0B00	5	* * * * - - -	Circular shift D by B bits (0=use R0)	SRC R5,4
SRL D,B	Shift Right Logical	0900	5	* * * * - - -		
AI D,nn	Add Immediate	0220	8	* * * * * - -	Add n to reg D	
ANDI D,nn	And Immediate	0240	8	* * * - - - -		AI R2,>FF
CI D,nn	Compare Immediate	0280	8	* * * - - - -	Compare D to n	CI R2,>10E
LI D,nn	Load Immediate	0200	8	* * * - - - -		
ORI	Or Immediate	0260	8	* * * - - - -		
LWPI A	Load Workspace Pointer Immediate	02E0	x	- - - - - - -	A→WP	LWPI >FCOO
LIMI	Load Interrupt Mask	0300	x	- - - - - - -		
STST	Store Status Register	02C0	x	- - - - - - -		
STWP	Store Workspace Pointer	02A0	x	- - - - - - -		STWP R2
RTWP	Return from Context Switch	0380	x	* * * * * * *	R13→WP, R14→PC, R15→ST	
IDLE	Idle	0340	7	- - - - - - -		
RSET	Reset	0360	7	- - - - - - -		
CKOF	User Defined	03C0	7	- - - - - - -		
CKON	User Defined	03A0	7	- - - - - - -		
LREX	User Defined	03E0	7	- - - - - - -		
B *R11	RETurn					

MIPS

	Instruction		Delay?	RISCV	Example		F E D C B A 9 8 7 6 5 4 3 2 1 0 F E D C B A 9 8 7 6 5 4 3 2 1 0
	LA dest,addr		*Load*	LA		Load address	LUI $at,>label ORI Rd,$at,<label
I	**LB** dest,addr	R3000	*Load*	LB		Load byte	1 0 0 0 0 0 s s s s s t t t t t i i i i i i i i i i i i i i i i
I	**LBU** dest,addr	R3000	*Load*	LBU		Load byte unsigned	1 0 0 1 0 0 s s s s s t t t t t i i i i i i i i i i i i i i i i
	LH dest,addr	R3000	*Load*	LH		Load halfword	1 0 0 0 0 1 s s s s s t t t t t i i i i i i i i i i i i i i i i
I	**LHU** dest,addr	R3000	*Load*	LHU		Load halfword unsigned	1 0 0 1 0 1 s s s s s t t t t t i i i i i i i i i i i i i i i i
I	**LW** dest,addr	R3000	*Load*	LW		Load word	1 0 0 0 1 1 s s s s s t t t t t i i i i i i i i i i i i i i i i
	LWL dest,addr	R3000				Load word left (can Load partial data from unword aligned data)	1 0 0 0 1 0 s s s s s t t t t t i i i i i i i i i i i i i i i i
	LWR dest,addr	R3000				Load word right (can Load partial data from unword aligned data)	1 0 0 1 1 0 s s s s s t t t t t i i i i i i i i i i i i i i i i
	LD dest,addr					Load double	lui $at,%hi_addr addiu $at,$at,%lo_addr lw $t2,0($at) lw $t3,4($at)
	ULH dest,addr					Unaligned Load Halfword	LB Rd,4(Rs) LBU $at,3(Rs) SLL Rd,Rd,8 OR Rd,Rd,$at
	ULHU dest,addr					Unaligned Load Halfword Unsigned	LBU Rd,4(Rs) LBU $at,3(Rs) SLL Rd,Rd,8 OR Rd,Rd,$at
	ULW dest,addr					Unaligned Load Word	LWL Rd, 6(Rs) LWR Rd,3(Rs)
	LI dest,expr			LI		Load Immediate	LUI $at,>imm ORI Rd,$at,<imm ori Rt,$0,imm
I	**LUI** dest,expr	R3000		LUI		Load Upper Immediate 0xFFFF----	0 0 1 1 1 1 0 0 0 0 0 t t t t t i i i i i i i i i i i i i i i i
I	**SB** source,addr	R3000		SB		Store Byte	1 0 1 0 0 0 s s s s s t t t t t i i i i i i i i i i i i i i i i
	SD expr,dest					Store Doubleword	
I	**SH** expr,dest	R3000		SH		Store Halfword	1 0 1 0 0 1 s s s s s t t t t t i i i i i i i i i i i i i i i i
	SWL expr,dest	R3000				Store Word Left (can Store partial data from unword aligned data)	1 0 1 0 1 0 s s s s s t t t t t i i i i i i i i i i i i i i i i
	SWR expr,dest	R3000				Store Word Right (can Store partial data from unword aligned data)	1 0 1 1 1 0 s s s s s t t t t t i i i i i i i i i i i i i i i i
I	**SW** expr,dest	R3000		SW		Store Word	1 0 1 0 1 1 s s s s s t t t t t i i i i i i i i i i i i i i i i
	USH dest,expr					Unaligned Store Half Word	SB Rd,3(Rs) SRI $at,Rd,8 SB $at,4(Rs)
	USW dest,expr					Unaligned Store Word	SWL Rd,6(Rs) SWR Rd,3(Rs)
R	**ADD** rd, rs, rt	R3000		ADD		Add (Signed)	0 0 0 0 0 0 s s s s s t t t t t d d d d d 0 0 0 0 0 1 0 0 0 0 0
I	**ADDI** rt,rs,imm	R3000		ADDI		Add Immediate	0 0 1 0 0 0 s s s s s t t t t t i i i i i i i i i i i i i i i i
I	**ADDIU** dest,src1,imm	R3000				Add Immediate Unsigned	0 0 1 0 0 1 s s s s s t t t t t i i i i i i i i i i i i i i i i
R	**ADDU** dest,src1,src2	R3000				Add Unsigned	0 0 0 0 0 0 s s s s s t t t t t d d d d d 0 0 0 0 0 1 0 0 0 0 1
R	**AND** rd,rs,rt	R3000		AND		AND	0 0 0 0 0 0 s s s s s t t t t t d d d d d 0 0 0 0 0 1 0 0 1 0 0
I	**ANDI** dest,src1,imm	R3000		ANDI		And Immediate	0 0 1 1 0 0 s s s s s t t t t t i i i i i i i i i i i i i i i i
R	**DIV** dest,src1,src2	R3000		DIV		Divide (Signed) (Low=Quotient / High=Remainder)	0 0 0 0 0 0 s s s s s t t t t t 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0
R	**DIVU** dest,src1,src2	R3000		DIVU		Divide Unsigned (Low=Quotient / High=Remainder)	0 0 0 0 0 0 s s s s s t t t t t 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1
R	**XOR** dest,src1,src2	R3000				Exclusive OR	0 0 0 0 0 0 s s s s s t t t t t d d d d d 0 0 0 0 0 0 1 0 1 1 0
	XORI dest,src,imm	R3000				Xor Immediate	0 0 1 1 1 0 s s s s s t t t t t i i i i i i i i i i i i i i i i
	MUL dest,src1,src2			MUL		Multiply	MULT Rs,Rt MFLO Rd
	MULO dest,src1,src2			MULH		Multiply with Overflow	MULT Rs,Rt MFHI $at MFLO Rd SRA Rd,Rd,31 BEQ $at,Rd,ok BREAK $0 ok:MFLO Rd
	MULOU dest,src1,src2			MULHU		Multiply with Overflow Unsigned	MULTU Rs,Rt MFHI $at BEQ $at, $0, ok ok: BREAK $0 MFLO Rd
R	**NOR** dest,src1,src2	R3000				Not Or	0 0 0 0 0 0 s s s s s t t t t t d d d d d 0 0 0 0 0 1 0 0 1 1 1
R	**OR** dest,src1,src2	R3000		OR		OR	0 0 0 0 0 0 s s s s s t t t t t d d d d d 0 0 0 0 0 1 0 0 1 0 1
I	**ORI** dest,src1,imm	R3000		ORI		OR Immediate	0 0 1 1 0 1 s s s s s t t t t t i i i i i i i i i i i i i i i i
	SEQ dest,src1,src2					Set Equal	BEQ Rt,Rs,yes ORI Rd,$0,0 BRQ $0,$0,skip yes:ORI Rd,$0,1 skip:
	SGT dest,src1,src2					Set Greater	SLT Rd, Rt, Rs
	SGE dest,src1,src2					Set Greater/Equal	BNE Rt,Rs,yes ORI Rd,$0,1 BEQ $0,$0,skip yes:SLT Rd,Rt,Rs skip:
	SGEU dest,src1,src2					Set Greater/Equal Unsigned	BNE Rt,Rs,yes ORI Rd,$0,1 BEQ $0,$0,skip yes:SITU Rd,Rt,Rs skip:
	SGTU dest,src1,src2					Set Greater Unsigned	SLTU Rd, Rt, Rs
R	**SLT** dest,src1,src2	R3000				Set Less Than	0 0 0 0 0 0 s s s s s t t t t t d d d d d 0 0 0 0 0 1 0 1 0 1 0
I	**SLTI** dest,src,imm	R3000				Set on Less Than Immediate	0 0 1 0 1 0 s s s s s t t t t t i i i i i i i i i i i i i i i i
I	**SLTIU** dest,src,imm	R3000				Set on Less Than Immediate Unsigned	0 0 1 0 1 1 s s s s s t t t t t i i i i i i i i i i i i i i i i
	SLE dest,src1,src2					Set Less/Equal	BNE Rt,Rs,yes ORI Rd,$0,1 BEQ $0,$0,skip yes:SLT Rd, Rs, Rt skip:
	SLEU dest,src1,src2					Set Less/Equal Unsigned	BNE Rt,Rs,yes ORI Rd,$0,1 BEQ $0,$0,skip yes:SLTU Rd,Rs,Rt skip:
R	**SLTU** dest,src1,src2	R3000				Set Less Unsigned	0 0 0 0 0 0 s s s s s t t t t t d d d d d 0 0 0 0 0 1 0 1 0 1 0
	SNE dest,src1,src2					Set Not Equal	BRQ Rt,Rs,yes ORI Rd,$0,1 BEQ $0,$0,skip yes:ORI Rd,$0,0 skip:
R	**SUB** dest,src1,src2	R3000				Subtract (With Overflow)	0 0 0 0 0 0 s s s s s t t t t t d d d d d 0 0 0 0 0 1 0 0 0 1 0
R	**SUBU** dest,src1,src2	R3000				Subtract (Without Overflow)	0 0 0 0 0 0 s s s s s t t t t t d d d d d 0 0 0 0 0 1 0 0 0 1 1
	REM dest,src1,src2			REM		Remainder (Signed)	BNE Rt,$0,8 BREAK $0 DIV Rs,Rt MFHI Rd
	REMU dest,src1,src2			REMU		Remainder (Unsigned)	BNE Rt,$0,ok BREAK $0 ok: DIVU Rs,Rt MFHI Rd
	ROL dest,src1,src2					Rotate Left (Reg or imm)	SUBU $at,$0,Rt SRLV $at,Rs,$at SLLV Rd,Rs,Rt OR Rd,Rd,$a SRL $at,Rs,32-sa SLL Rd,Rs,sa OR Rd,Rd,$at
	ROR dest,src1,src2					Rotate Right (Reg or imm)	SUBU $at,$0,Rt SLLV $at,Rs,$at SRLV Rd,Rs,Rt OR Rd,Rd,$at SLL $at,Rs,32-sa SRI Rd,Rs,sa OR Rd,Rd,$at
	SRA dest,src1,imm	R3000		SRAI		Shift Right Arithmatic Immediate	0 0 0 0 0 0 0 0 0 0 0 t t t t t d d d d d i i i i i 0 0 0 0 1 1
R	**SRAV** dest,src1,src2	R3000		SRA		Shift Right Arithmatic	0 0 0 0 0 0 s s s s s t t t t t d d d d d 0 0 0 0 0 0 0 0 1 1 1
	SLL dest,src, imm	R3000		SLLI		Shift Left Logical Immediate	0 0 0 0 0 0 0 0 0 0 0 t t t t t d d d d d i i i i i 0 0 0 0 0 0
R	**SLLV** dest,src1,src2	R3000		SLL		Shift Left Logical by Variable	0 0 0 0 0 0 s s s s s t t t t t d d d d d 0 0 0 0 0 0 0 1 0 0 0
	SRL dest,src, imm	R3000		SRLI		Shift Right Logical Immediate	0 0 0 0 0 0 0 0 0 0 0 t t t t t d d d d d i i i i i 0 0 0 0 1 0
R	**SRLV** dest,src1,src2	R3000		SRL		Shift Right Logical by Variable	0 0 0 0 0 0 s s s s s t t t t t d d d d d 0 0 0 0 0 0 0 0 1 1 0
	ABS dest,src					Absolute value	ADDU Rd,$0,Rs BGEZ Rs,1 SUB Rd,$0,Rs
	NEG dest,src			NEG		Negate (Signed)	SUB Rd,$0,Rs
	NEGU dest,src					Negate (Unsigned)	SUBU Rd,$0,Rs
	NOT dest,src			NOT		Not Or	NOR Rd,Rs,$0
	MOVE dest,src			MV		Move	ADDU Rd,$0,Rs
R	**MULT** src1,src2	R3000				Multiply... result in HI/LOW (leave 2 instructions before next Mult/Div)	0 0 0 0 0 0 s s s s s t t t t t 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0
R	**MULTU** src1,src2	R3000				Multiply (Unsigned)... result in HI/LOW (leave 2 instructions before next Mult/Div)	0 0 0 0 0 0 s s s s s t t t t t 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1
J	**J** addr	R3000		J		Jump	0 0 0 0 1 0 i i i i i i i i i i i i i i i i i i i i i i i i i i
J	**JAL** addr	R3000		JAL		Jump and link	0 0 0 0 1 1 i i i i i i i i i i i i i i i i i i i i i i i i i i
	JALR return,reg	R3000		JA:LR		Jump and link Register	0 0 0 0 0 0 s s s s s 0 0 0 0 0 d d d d d 0 0 0 0 0 0 0 1 0 0 1
R	**JR** reg	R3000		JR		Jump to address in register	0 0 0 0 0 0 s s s s s 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
I	**BEQ** src1, src2, label	R3000	***Branch***	BEQ		Branch on Equal	0 0 0 1 0 0 s s s s s t t t t t i i i i i i i i i i i i i i i i
	BGT src1, src2, label		***Branch***			Branch on Greater	SLT $at,Rs,Rt BNE $at,$0,Label
	BGE src1, src2, label		***Branch***	BGE		Branch on Greater/Equal	SLT $at, rs,rt BEQ $at,$0,Label
	BGEU src1, src2, label		***Branch***	BGEU		Branch on Greater/Equal unsigned	SLTU $at, rs,rt BEQ $at,$0,Label
	BGTU src1, src2, label		***Branch***			Branch on Greater unsigned	SLTU $at,Rs,Rt BNE $at,$0,Label
	BLT src1, src2, label		***Branch***	BLT		Branch on Less than	SLT $at,Rs,Rt BNE $at,$0,Label
	BLE src1, src2, label		***Branch***			Branch on Less/Equal	SLT $at, Rt,Rs BEQ $at,$0,Label
	BLEU src1, src2, label		***Branch***			Branch on Less/Equal Unsigned	SLTU $at, Rt,Rs BEQ $at,$0,Label
	BLTU src1, src2, label		***Branch***	BLTU		Branch on Less Unsigned	SLTU $at,Rs,Rt BNE $at,$0,Label
I	**BNE** src1, src2, label	R3000	***Branch***	BNE		Branch on Not Equal	0 0 0 1 0 1 s s s s s t t t t t i i i i i i i i i i i i i i i i
	BEQZ src1, label					Branch on Equal to Zero	BEQ Rs,$0,Label
	BGEZ src1, label	R3000				Branch on greater/equal to zero	0 0 0 0 0 1 s s s s s 0 0 0 0 1 i i i i i i i i i i i i i i i i
I	**BGTZ** src1, label					Branch on Greater than zero	0 0 0 1 1 1 s s s s s 0 0 0 0 0 i i i i i i i i i i i i i i i i
	BGEZAL src1, label	R3000				Branch on Greater or equal to zero	0 0 0 0 0 1 s s s s s 1 0 0 0 1 i i i i i i i i i i i i i i i i
	BLTZAL src1, label	R3000				Branch on less than zero and link	0 0 0 0 0 1 s s s s s 1 0 0 0 0 i i i i i i i i i i i i i i i i
I	**BLEZ** src1, label	R3000				Branch on Less than or equal to zero	0 0 0 1 1 0 s s s s s 0 0 0 0 0 i i i i i i i i i i i i i i i i
	BLTZ src1, label	R3000				Branch on less than zero	0 0 0 0 0 1 s s s s s 0 0 0 0 0 i i i i i i i i i i i i i i i i
	BNEZ src1, label					Branch on not equal to zero	BNE Rs,$0,Label
	B label					Branch	Bgez $0,label
	BAL label					Branch and Link	0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 i i i i i i i i i i i i i i i i
	BREAK breakcode	R3000				Break	0 0 0 0 0 0 i i i i i i i i i i i i i i i i i i i i i i 0 0 1 1 0 1
	RFE					Restore from exception	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
	SYSCALL			ecall		System Call	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 i i i i i i i i
R	**MFHI** register	R3000				Move from HI to Register	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 d d d d d 0 0 0 0 0 0 1 0 0 0 0
R	**MTHI** register	R3000				Move to HI from Register	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 d d d d d 0 0 0 0 0 0 1 0 0 0 1
R	**MFLO** register	R3000				Move from LOW to register	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 d d d d d 0 0 0 0 0 0 1 0 0 1 0
R	**MTLO** register	R3000				Move to LOW to Register	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 d d d d d 0 0 0 0 0 0 1 0 0 1 1
	LWCz dest,addr	R3000			LWC2 $1,0(a0)	Load Word Coprocessor Z	1 1 0 0 0 1 b b b b b r r r r r i i i i i i i i i i i i i i i i
	SWCz source,addr	R3000			SWC2 $1,0(a0)	Store Word Coprocessor z	1 1 1 0 0 1 b b b b b f f f f f i i i i i i i i i i i i i i i i
R	**MFCz** dest-gpr,source	R3000			MFC2 a0,$1	Move from Coprocessor z	0 1 0 0 0 0 0 0 0 0 0 t t t t t d d d d d 0 0 0 0 0 0 0 0 0 0 0
	MTCz src-gpr, destination	R3000			MTC2 a0,$1	Move to Coprocessor z	0 1 0 0 0 0 0 0 1 0 0 t t t t t d d d d d 0 0 0 0 0 0 0 0 0 0 0
	CFCz dest-gpr, source	R3000			CFC2 a0,$1	control from Coprocessor z	
	COPz cofun	R3000			COP2 1	Perform Coprocessor operation cofun on coprocessor z	
	CTCz src-gpr,destionation	R3000			CTC2 a0,$1	move reg RT into control register RD of coprocessor z	
	NOP					No Operation	OR $0,$0,$0
	BCzF label					Branch Coprocessor z false	
	BCzT label					Branch Coprocessor z true	
	Cz expr		***Store***			Coprocessor z operation	

Risc-V

	Group	Instruction	MIPS	Name	Pseudocode	F E D C B A 9 8 7 6 5 4 3 2 1 0 F E D C B A 9 8 7 6 5 4 3 2 1 0					
U	RV32I	**LUI** rd,imm	LUI	Load Upper Immediate (Top 20 bits of rd)	rd ← imm	Imm [31:12]				rd	0010111
U	RV32I	**AUIPC** rd,offset		Add Upper Immediate to PC	rd ← pc + offset	Imm [31:12]				rd	0010111
UJ	RV32I	**JAL** rd,offset	JAL	Jump and Link	rd ← pc + len(inst)··· pc ← pc + off	[20][10:1]	[11][19:12]			rd	1101111
I	RV32I	**JALR** rd,rs1,offset	JALR	Jump and Link Register (return to rs1 when off=0)	rd ← pc + len(inst)···pc ← (rs1+off)∧−2	Imm [11:0]		Rs1	000	rd	1100111
SB	RV32I	**BEQ** rs1,rs2,offset	BEQ	Branch Equal	if rs1 = rs2 then pc ← pc + offset	Imm [12][10:0]	Rs2	Rs1	000	Im [11][4:1]	1100011
SB	RV32I	**BGE** rs1,rs2,offset	BGE	Branch Greater than Equal	if rs1 ≥ rs2 then pc ← pc + offset	Imm [12][10:3]	Rs2	Rs1	101	Im [11][4:1]	1100011
SB	RV32I	**BGEU** rs1,rs2,offset	BGEU	Branch Greater than Equal Unsigned	if rs1 ≥ rs2 then pc ← pc + offset	Imm [12][10:5]	Rs2	Rs1	111	Im [11][4:1]	1100011
SB	RV32I	**BLT** rs1,rs2,offset	BLT	Branch Less Than	if rs1 < rs2 then pc ← pc + offset	Imm [12][10:2]	Rs2	Rs1	100	Im [11][4:1]	1100011
SB	RV32I	**BLTU** rs1,rs2,offset	BLTU	Branch Less Than Unsigned	if rs1 < rs2 then pc ← pc + offset	Imm [12][10:4]	Rs2	Rs1	110	Im [11][4:1]	1100011
SB	RV32I	**BNE** rs1,rs2,offset	BNE	Branch Not Equal	if rs1 ≠ rs2 then pc ← pc + offset	Imm [12][10:1]	Rs2	Rs1	001	Im [11][4:1]	1100011
I	RV32I	**LB** rd,offset(rs1)	LB	Load Byte	rd ← s8[rs1 + offset]	Imm [11:0]		Rs1	000	rd	0000011
I	RV32I	**LBU** rd,offset(rs1)	LBU	Load Byte Unsigned	rd ← u8[rs1 + offset]	Imm [11:0]		Rs1	100	rd	0000011
I	RV32I	**LH** rd,offset(rs1)	LH	Load Half	rd ← s16[rs1 + offset]	Imm [11:0]		Rs1	001	rd	0000011
I	RV32I	**LHU** rd,offset(rs1)	LHU	Load Half Unsigned	rd ← u16[rs1 + offset]	Imm [11:0]		Rs1	101	rd	0000011
I	RV32I	**LW** rd,offset(rs1)	LW	Load Word	rd ← s32[rs1 + offset]	Imm [11:0]		Rs1	010	rd	0000011
S	RV32I	**SB** rs2,offset(rs1)	SB	Store Byte	u8[rs1 + offset] ← rs2	Imm [11:5]	Rs2	Rs1	000	Imm [4:0]	0100011
S	RV32I	**SH** rs2,offset(rs1)	SH	Store Half	u16[rs1 + offset] ← rs2	Imm [11:5]	Rs2	Rs1	001	Imm [4:0]	0100011
S	RV32I	**SW** rs2,offset(rs1)	SW	Store Word	u32[rs1 + offset] ← rs2	Imm [11:5]	Rs2	Rs1	010	Imm [4:0]	0100011
I	RV32I	**ADDI** rd,rs1,imm	ADDI	Add Immediate	rd ← rs1 + sx(imm)	Imm [11:0]		Rs1	000	rd	0010011
I	RV32I	**ANDI** rd,rs1,imm	ANDI	And Immediate	rd ← ux(rs1) ∧ ux(imm)	Imm [11:0]		Rs1	111	rd	0010011
I	RV32I	**ORI** rd,rs1,imm		Or Immediate	rd ← ux(rs1) ∨ ux(imm)	Imm [11:0]		Rs1	110	rd	0010011
I	RV32I	**SLTI** rd,rs1,imm	SLTI	Set Less Than Immediate (rd=1/0)	rd ← sx(rs1) < sx(imm)	Imm [11:0]		Rs1	010	rd	0010011
I	RV32I	**SLTIU** rd,rs1,imm	SLTIU	Set Less Than Immediate Unsigned (rd=1	rd ← ux(rs1) < ux(imm)	Imm [11:0]		Rs1	011	rd	0010011
I	RV32I	**XORI** rd,rs1,imm	XORI	Xor Immediate	rd ← ux(rs1) ⊕ ux(imm)	Imm [11:0]		Rs1	100	rd	0010011
R	RV32I	**SLLI** rd,rs1,imm		Shift Left Logical Immediate	rd ← ux(rs1) ≪ ux(imm)	0000000	shamt	Rs1	001	rd	0010011
R	RV32I	**SRLI** rd,rs1,imm		Shift Right Logical Immediate	rd ← ux(rs1) ≫ ux(imm)	0000000	shamt	Rs1	101	rd	0010011
R	RV32I	**SRAI** rd,rs1,imm		Shift Right Arithmetic Immediate	rd ← sx(rs1) ≫ ux(imm)	0100000	shamt	Rs1	101	rd	0010011
S	RV32I	**ADD** rd,rs1,rs2	ADD	Add	rd ← sx(rs1) + sx(rs2)	0000000	shamt	Rs1	000	rd	0110011
S	RV32I	**AND** rd,rs1,rs2		And	rd ← ux(rs1) ∧ ux(rs2)	0000000	shamt	Rs1	111	rd	0110011
S	RV32I	**OR** rd,rs1,rs2		Or	rd ← ux(rs1) ∨ ux(rs2)	0000000	shamt	Rs1	110	rd	0110011
S	RV32I	**SUB** rd,rs1,rs2		Subtract	rd ← sx(rs1) − sx(rs2)	0100000	shamt	Rs1	000	rd	0110011
S	RV32I	**SLL** rd,rs1,rs2	SLLV	Shift Left Logical	rd ← ux(rs1) ≪ rs2	0000000	shamt	Rs1	001	rd	0110011
S	RV32I	**SLT** rd,rs1,rs2		Set Less Than	rd ← sx(rs1) < sx(rs2)	0000000	shamt	Rs1	010	rd	0110011
S	RV32I	**SLTU** rd,rs1,rs2		Set Less Than Unsigned	rd ← ux(rs1) < ux(rs2)	0000000	shamt	Rs1	011	rd	0110011
S	RV32I	**SRL** rd,rs1,rs2	SRLV	Shift Right Logical	rd ← ux(rs1) ≫ rs2	0000000	shamt	Rs1	101	rd	0110011
S	RV32I	**SRA** rd,rs1,rs2	SRAV	Shift Right Arithmetic	rd ← sx(rs1) ≫ rs2	0100000	shamt	Rs1	101	rd	0110011
S	RV32I	**XOR** rd,rs1,rs2	XOR	Xor	rd ← ux(rs1) ⊕ ux(rs2)	0000000	shamt	Rs1	100	rd	0110011
I	RV32I	**FENCE** pred,succ		Fence		Fm Pred	Suc	Rs1	000	rd	1110011
I	RV32I	**FENCE.I**		Fence Instruction		Imm		Rs1	001	rd	0001111
I	RV64I	**LWU** rd,offset(rs1)		Load Word Unsigned	rd ← u32[rs1 + offset]	Imm [11:0]		Rs1	110	rd	0000011
	RV64I	**LD** rd,offset(rs1)		Load Double	rd ← u64[rs1 + offset]	Imm [11:0]		Rs1	011	rd	0000011
	RV64I	**SD** rs2,offset(rs1)		Store Double	u64[rs1 + offset] ← rs2	Imm [11:5]	Rs2	Rs1	011	Imm [4:0]	0100011
	RV64I	**SLLI** rd,rs1,imm	SLL	Shift Left Logical Immediate	rd ← ux(rs1) ≪ sx(imm)	000000	shamt	Rs1	001	rd	0010011
	RV64I	**SRLI** rd,rs1,imm	SRL	Shift Right Logical Immediate	rd ← ux(rs1) ≫ sx(imm)	000000	shamt	Rs1	101	rd	0010011
	RV64I	**SRAI** rd,rs1,imm	SRA	Shift Right Arithmetic Immediate	rd ← sx(rs1) ≫ sx(imm)	010000	shamt	Rs1	101	rd	0010011
	RV64I	**ADDIW** rd,rs1,imm		Add Immediate Word	rd ← s32(rs1) + imm	Imm [11:0]		Rs1	000	rd	0011011
	RV64I	**SLLIW** rd,rs1,imm		Shift Left Logical Immediate Word	rd ← s32(u32(rs1) ≪ imm)	000000	shamt	Rs1	001	rd	0011011
	RV64I	**SRLIW** rd,rs1,imm		Shift Right Logical Immediate Word	rd ← s32(u32(rs1) ≫ imm)	000000	shamt	Rs1	101	rd	0011011
	RV64I	**SRAIW** rd,rs1,imm		Shift Right Arithmetic Immediate Word	rd ← s32(rs1) ≫ imm	010000	shamt	Rs1	101	rd	0011011
	RV64I	**ADDW** rd,rs1,rs2		Add Word	rd ← s32(rs1) + s32(rs2)	000000	Rs2	Rs1	000	rd	0111011
	RV64I	**SUBW** rd,rs1,rs2		Subtract Word	rd ← s32(rs1) − s32(rs2)	010000	Rs2	Rs1	000	rd	0111011
	RV64I	**SLLW** rd,rs1,rs2		Shift Left Logical Word	rd ← s32(u32(rs1) ≪ rs2)	000000	Rs2	Rs1	001	rd	0111011
	RV64I	**SRLW** rd,rs1,rs2		Shift Right Logical Word	rd ← s32(u32(rs1) ≫ rs2)	000000	Rs2	Rs1	101	rd	0111011
	RV64I	**SRAW** rd,rs1,rs2		Shift Right Arithmetic Word	rd ← s32(rs1) ≫ rs2	010000	Rs2	Rs1	101	rd	0111011
nn	RV32M	**MUL** rd,rs1,rs2	MULT	Multiply (First 32 bits)	rd ← ux(rs1) × ux(rs2)	0000001	Rs2	Rs1	000	Rd	0110011
S	RV32M	**MULH** rd,rs1,rs2		Multiply High (Second 32 bits)	rd ← (sx(rs1) × sx(rs2)) ≫ xlen	0000001	Rs2	Rs1	001	Rd	0110011
S	RV32M	**MULHSU** rd,rs1,rs2		Multiply High Signed*Unsigned Mix	rd ← (sx(rs1) × ux(rs2)) ≫ xlen	0000001	Rs2	Rs1	010	Rd	0110011
S	RV32M	**MULHU** rd,rs1,rs2		Multiply High Unsigned	rd ← (ux(rs1) × ux(rs2)) ≫ xlen	0000001	Rs2	Rs1	011	Rd	0110011
S	RV32M	**DIV** rd,rs1,rs2	DIV	Divide Signed	rd ← sx(rs1) ÷ sx(rs2)	0000001	Rs2	Rs1	100	Rd	0110011
S	RV32M	**DIVU** rd,rs1,rs2	DIVU	Divide Unsigned	rd ← ux(rs1) ÷ ux(rs2)	0000001	Rs2	Rs1	101	Rd	0110011
S	RV32M	**REM** rd,rs1,rs2	REM	Remainder Signed	rd ← sx(rs1) mod sx(rs2)	0000001	Rs2	Rs1	110	Rd	0110011
S	RV32M	**REMU** rd,rs1,rs2	REMU	Remainder Unsigned	rd ← ux(rs1) mod ux(rs2)	0000001	Rs2	Rs1	111	Rd	0111011
S	RV64M	**MULW** rd,rs1,rs2		Multiple Word	rd ← u32(rs1) × u32(rs2)	0000001	Rs2	Rs1	000	Rd	0111011
S	RV64M	**DIVW** rd,rs1,rs2		Divide Signed Word	rd ← s32(rs1) ÷ s32(rs2)	0000001	Rs2	Rs1	100	Rd	0111011
S	RV64M	**DIVUW** rd,rs1,rs2		Divide Unsigned Word	rd ← u32(rs1) ÷ u32(rs2)	0000001	Rs2	Rs1	101	Rd	0111011
S	RV64M	**REMW** rd,rs1,rs2		Remainder Signed Word	rd ← s32(rs1) mod s32(rs2)	0000001	Rs2	Rs1	110	Rd	0111011
S	RV64M	**REMUW** rd,rs1,rs2		Remainder Unsigned Word	rd ← u32(rs1) mod u32(rs2)	0000001	Rs2	Rs1	111	Rd	0111011
	Directive	**.2byte**		16-bit comma separated words (unaligned)							
	Directive	**.4byte**		32-bit comma separated words (unaligned)							
	Directive	**.8byte**		64-bit comma separated words (unaligned)							
	Directive	**.half**		16-bit comma separated words (naturally aligned)							
	Directive	**.word**		32-bit comma separated words (naturally aligned)							
	Directive	**.dword**		64-bit comma separated words (naturally aligned)							
	Directive	**.byte**		8-bit comma separated words							
	Directive	**.dtpreldword**		64-bit thread local word							
	Directive	**.dtprelword**		32-bit thread local word							
	Directive	**.sleb128** expression		signed little endian base 128, DWARF							
	Directive	**.uleb128** expression		unsigned little endian base 128, DWARF							
	Directive	**.asciz** “string”		emit string (alias for .string)							
	Directive	**.string**		emit string							
	Directive	**.incbin** “filename”		emit the included file as a binary sequence of octets							
	Directive	**.zero** integer		zero bytes							
	Directive	**.align** integer		align to power of 2 (alias for .p2align)							
	Directive	**.balign** b,[pad_val=0]		byte align							
	Directive	**.p2align** p2,[pad_val=0],max		align to power of 2							
	Directive	**.globl** symbol_name		emit symbol_name to symbol table (scope GLOBAL)							
	Directive	**.local** symbol_name		emit symbol_name to symbol table (scope LOCAL)							
	Directive	**.equ** name, value		constant definition							
	Directive	**.text**		emit .text section (if not present) and make current							
	Directive	**.data**		emit .data section (if not present) and make current							
	Directive	**.rodata**		emit .rodata section (if not present) and make current							
	Directive	**.bss**		emit .bss section (if not present) and make current							

Directive	**.comm** sym_nam,sz,aln		emit common object to .bss section	
Directive	**.common** sym_name,sz,aln		emit common object to .bss section	
Directive	**.section** sect		emit section (if not present, default .text	[{.text,.data,.rodata,.bss}]
Directive	**.option** opt		RISC-V options	{rvc,norvc,pic,nopic,push,pop}
Directive	**.macro** name arg1 [, argn]		begin macro definition ¥argname to substitute	
Directive	**.endm**		end macro definition	
Directive	**.file** “filename”		emit filename FILE LOCAL symbol table	
Directive	**.ident** “string”		accepted for source compatibility	
Directive	**.size** symbol, symbol		accepted for source compatibility	
Directive	**.type** symbol, @function		accepted for source compatibility	
Psuedo	**NOP**		No operation	addi zero,zero,0
Psuedo	**LI** rd, imm	LI	Load immediate	(several expansions) (LUA+ADDI)
Psuedo	**LA** rd, symbol	LA	Load address	(several expansions)
Psuedo	**MV** rd, rs1	MOVE	Copy register	addi rd, rs, 0
Psuedo	**NOT** rd, rs1	NOT	One’s complement	xori rd, rs, -1
Psuedo	**NEG** rd, rs1	NEG	Two’s complement	sub rd, x0, rs
Psuedo	NEGW rd, rs1		Two’s complement Word	subw rd, x0, rs
Psuedo	SEXT.W rd, rs1		Sign extend Word	addiw rd, rs, 0
Psuedo	**SEQZ** rd, rs1		Set if = zero	sltiu rd, rs, 1
Psuedo	**SNEZ** rd, rs1		Set if ≠ zero	sltu rd, x0, rs
Psuedo	**SLTZ** rd, rs1		Set if < zero	slt rd, rs, x0
Psuedo	**SGTZ** rd, rs1		Set if > zero	slt rd, x0, rs
Psuedo	FMV.S frd, frs1		Single-precision move	fsgnj.s frd, frs, frs
Psuedo	FAB.S frd, frs1		Single-precision absolute value	fsgnjx.s frd, frs, frs
Psuedo	FNEG.S frd, frs1		Single-precision negate	fsgnjn.s frd, frs, frs
Psuedo	FMV.D frd, frs1		Double-precision move	fsgnj.d frd, frs, frs
Psuedo	FABS.D frd, frs1		Double-precision absolute value	fsgnjx.d frd, frs, frs
Psuedo	FNEG.D frd, frs1		Double-precision negate	fsgnjn.d frd, frs, frs
Psuedo	**BEQZ** rs1, offset		Branch if = zero	beq rs, x0, offset
Psuedo	**BNEZ** rs1, offset		Branch if ≠ zero	bne rs, x0, offset
Psuedo	**BLEZ** rs1, offset		Branch if ≤ zero	bge x0, rs, offset
Psuedo	**BGEZ** rs1, offset		Branch if ≥ zero	bge rs, x0, offset
Psuedo	**BLTZ** rs1, offset		Branch if < zero	blt rs, x0, offset
Psuedo	**BGTZ** rs1, offset		Branch if > zero	blt x0, rs, offset
Psuedo	**BGT** rs, rt, offset		Branch if >	blt rt, rs, offset
Psuedo	**BLE** rs, rt, offset		Branch if ≤	bge rt, rs, offset
Psuedo	**BGTU** rs, rt, offset		Branch if >, unsigned	bltu rt, rs, offset
Psuedo	**BLEU** rs, rt, offset		Branch if ≤, unsigned	bltu rt, rs, offset
Psuedo	**J** offset	J	Jump	jal x0, offset
Psuedo	**JR** offset	JR	Jump register	jal x1, offset
Psuedo	**RET**		Return from subroutine	jalr x0, x1, 0

Syntax: Imm [12][10:5] = Bits 12 & 10-5 of immediate (other bits in other part)

ARM

Mnemonic	Description	Example
ADCccS Rn, Rm, Op2	Add With Carry.	ADC R0,R0,#4
ADDccS Rn, Rm, Op2	Add Op2 to Rm and store the result in Rn.	ADD R0,R0,#4
ANDccS Rn, Rm, Op2	Logically AND Op2 with Rm and store the result in Rn.	AND R0,R0,#4
Bcc Label	Branch to a relative Label.	BEQ ConditionalJump
BICccS Rn, Rm, Op2	Logically Bit Clear Op2 with Rm and store the result in Rn.	BIC R0,R0,#4
BLcc Label	Branch and Link to a relative subroutine Label.	BL TestSub
CMNcc Rn, Op2	Compare Negative Rn to Op2. set the flags like"ADDS Rn,Op2"	CMN R0,#4
CMPcc Rn, Op2	Compare Rn to Op2. set the flags, the same as "SUBS Rn,Op2"	CMP R0,#4
EORccS Rn, Rm, Op2	Logically Exclusive OR Op2 with Rm and store the result in Rn.	EOR R0,R0,#4
LDMccadm Rn!, {Regs}	Transfer range of registers {Regs} to address in Rn. Like POP	LDMFD sp!,{r0,r1,r2}
LDRcc Rn, Flex **LDRccB Rn, Flex**	Load register Rn from address Flex	LDR R0,NearLabel
LDRccH Rn, Off **LDRccSH Rn, Off** **LDRccSB Rn, Off**	HalfWord (16 bit), Signed Word (16 Bit) and Signed Byte (8 Bit) load	LDRSB R0,[R1,#-255]
MLAccS Rn, Rm, Ro, Rp	32 bit Multiplication and Add. Rn=(Rm*Ro)+ Rp	MLA R0,R1,R2,R3
MOVccS Rn, Op2	Move value in Op2 into Rn.	MOV R0,#0xFF
MRScc Rn,sr	Move sr (either CPSR or SPSR) to register Rn.	MRS R0,SPSR
MSRcc sr_f,# **MSRcc sr_f,Rn**	Move immediate # or register into flags f of sr (either CPSR or SPSR).	MSR CPSR_F,#0
MULccS Rn, Rm, Ro	32 bit Multiplication. Rn=Rm*Ro.	MUL R0,R1,R2
MVNccS Rn, Op2	Move Not. Flip all the bits of Op2 and move result into Rn.	MVN R0,#0xFF
ORRccS Rn, Rm, Op2	Logically OR Op2 with Rm and store the result in Rn.	ORR R0,R0,#4
RSBccS Rn, Rm, Op2	Reverse Subtract. This performs the calculation Rn=Op2-Rm.	RSB R0,R0,#6
RSCccS Rn, Rm, Op2	Reverse Subtract with Carry. Rn=(Op2-Rm)-C .	RSC R0,R0,#6
SBCccS Rn, Rm, Op2	Reverse Subtract with Carry. Rn=(Op2-Rm)-C .	SBC R0,R0,#6
STMccadm Rn!, {Regs}	Transfer range of registers {Regs} to the address in Rn. Like PUSH	STMFD sp!,{r0,r1,r2}
STRcc Rn, Flex **STRccB Rn, Flex**	Store register Rn to address Flex.	STR r0,[r1,r2,asl #2]
STRccH Rn, Off **STRccSH Rn, Off** **STRccSB Rn, Off**	Half Word (16 bit), Signed half Word (16 Bit) and Signed Byte (8 Bit) store	STRSB R0,[R1,#-255]
SUBccS Rn, Rm, Op2	Subtract. This performs the calculation Rn=Rm-Op2.	SUB R0,R0,#6
SWIcc #	Software Interrupt.	SWI 3
SWPccB Rn, Rm, [Ro]	Swap a register and memory. Rn=[Ro], [Ro]=Rm.	SWPB R0,R1,[R2]
TEQcc Rn, Rm, Op2	Test for bitwise Equality. Set the flags like "EOR Rn,Rm,Op2"	TEQ R0,R0,#6
TSTcc Rn, Rm, Op2	Test bits. Set the flags like “AND Rn,Rm,Op2"	TST R0,R0,#6

Abbreviation	Meaning	Flag
EQ	EQual	Z=1
NE	Not Equal	Z=0
CS **HS**	Carry Set Higher or Same (Unsigned)	C=1
CC LO	Carry Clear LOwer (Unsigned)	C=0
MI	MInus (Negative)	N=1
PL	PLus (Positive)	N=0
VS	oVerflow Set	V=1
VC	oVerflow Clear	V=0
HI	HIgher (Unsigned)	C=1 and Z=0
LS	Lower or Same (Unsigned)	C=0 and Z=1
GE	Greater or Equal (Signed)	N=V
LT	Less Than (Signed)	N<>V
GT	Greater Than (Signed)	Z=0 and N=V
LE	Less than or Equal (Signed)	Z=1 or N<>V
AL	ALways	No condition

ARM Thumb

Command	Detail	Example	OP	Cycles	Opcode	N Z C V	ValidRegs
LDR Rd,[Rn,#]	LoaD Register (32 bit)	LDR r3,[r5,#0]				- - - -	R0-R7,#= 0 to 124 (Multiples of 4)
LDRB Rd,[Rn,#]	LoaD Register (8 bit)	LDRB r3,[r5,#2]				- - - -	R0-R7,#= 0 to 31 (Multiples of 1)
LDSB Rd,[Rn,#]	LoaD Register (Signed 8 bit)	LDRSB r3,[r5,#2]				- - - -	R0-R7,#= 0 to 31 (Multiples of 1)
LDRH Rd,[Rn,#]	LoaD Register (16 bit)	LDRH r3,[r5,#4]				- - - -	R0-R7,#= 0 to 62 (Multiples of 2)
LDSH Rd,[Rn,#]	LoaD Register (Signed 16 bit)	LDRSH r3,[r5,#4]				- - - -	R0-R7,#= 0 to 62 (Multiples of 2)
STR Rd,[Rn,#]	Store Register (32 Bit)	STR r3,[r5,#0]				- - - -	R0-R7,#= 0 to 124 (Multiples of 4)
STRB Rd,[Rn,#]	Store Register (8 Bit)	STRB r3,[r5,#0]				- - - -	R0-R7,#= 0 to 31 (Multiples of 1)
STRH Rd,[Rn,#]	Store Register (16 Bit)	STRH r3,[r5,#0]				- - - -	R0-R7,#= 0 to 62 (Multiples of 2)
POP {reglist}	Pop registers from the stack	POP {r1-r3,r5}				- - - -	R0-R7,LR
PUSH {reglist}	Push registers on to the stack	PUSH {r1-r3,r5}				- - - -	R0-R7,PC
LDMIA Rn!,{reglist}	Load Multiple and increment afte	LDMIA R0!,{r1-r3,r5}				- - - -	R0-R7
STMIA Rn!,{reglist}	Store Multiple and increment afte	STMIA R0!,{r1-r3,r5}				- - - -	R0-R7
ADD Rd,Rn,Rm	Add		Rd=Rn+Rm			N Z C V	R0-R7
ADD Rd,Rn,#	Add		Rd=Rn+#			N Z C V	R0-R7 #=0 to 7
ADD Rd,#	Add		Rd=Rd+#			N Z C V	R0-R7 #=0 to 255
SUB Rd,Rn,Rm	Subtract		Rd=Rn-Rm			N Z C V	R0-R7
SUB Rd,Rn,#	Subtract		Rd=Rn-#			N Z C V	R0-R7 #=0 to 7
SUB Rd,#	Subtract		Rd=Rd-#			N Z C V	R0-R7 #=0 to 255
ADD Rd,Rm	Add Low/High Regs (Can't both be low)		Rd=Rd+Rm			N Z C V	R0-R15,SP
ADD SP,#	Add to Stack Pointer		SP=SP+#			N Z C V	#0 to 508 (Multiple of 4)
SUB SP,#	Add to Stack Pointer		SP=SP+#			N Z C V	#0 to 508 (Multiple of 4)
ADD Rd,PC/SP,#	Add immediate to SP/PC		Rd=PC/SP+#			N Z C V	R0-R7, Rp=PC/SP #=0 to 1020 (Multiples of 4)
ADC Rd,Rm	Add with carry		Rd=Rd+Rm+C			N Z C V	R0-R7
SBC Rd,Rm	Subtract with carry		Rd=Rd-(Rm+C)			N Z C V	R0-R7
MUL Rd,Rm	Multiply		Rd=Rd*Rm			N Z - -	R0-R7
AND Rd,Rm	Logical AND		Rd=Rd AND Rm			N Z - -	R0-R7
ORR Rd,Rm	Logical OR		Rd=Rd OR Rm			N Z - -	R0-R7
EOR Rd,Rm	Logical Exclusive OR (XOR)		Rd=Rd EOR Rm			N Z - -	R0-R7
BIC Rd,Rm	Logical Bit Clear		Rd=Rd AND (NOT Rm)			N Z - -	R0-R7
ASR Rd,Rs	Arithmetic Shift Right Rs bits		Rd=Rd ASR Rs			N Z C -	R0-R7
ASR Rd,#	Arithmetic Shift Right # bits		Rd=Rd ASR #			N Z C -	R0-R7, #1 to 32
LSR Rd,Rs	Logical Shift Right Rs bits		Rd=Rd LSR Rs			N Z C -	R0-R7
LSR Rd,#	Logical Shift Right # bits		Rd=Rd LSR #			N Z C -	R0-R7, #1 to 32
LSL Rd,Rs	Logical Shift Left Rs bits		Rd=Rd LSL Rs			N Z C -	R0-R7
LSL Rd,#	Logical Shift Left # bits		Rd=Rd LSL #			N Z C -	R0-R7, #0 to 31
ROR Rd,Rs	Rotate Right Rs bits		Rd=Rd ROR Rs			N Z C -	R0-R7
CMP Rn,Rm	Compare (Set flags like SUB)		Flags=Rn-Rm			N Z C V	R0-R15
CMP Rn,#	Compare (Set flags like SUB)		Flags=Rn-#			N Z C V	R0-R7, #0 to 255
CMN Rn,Rm	Compare Negative (Set flags like ADD)		Flags=Rn+Rm			N Z C V	R0-R7
MOV Rd,#	Move Immediate		Rd=#			N Z C V	R0-R7, #0 to 255
MOV Rd,Rm	Move		Rd=Rm			N Z C V	R0-R15 (Flags unchanged R8+)
MVN Rd,Rm	Move Not (Flip bits of Rm)		Rd=NOT Rm			N Z C V	R0-R7
NEG Rd,Rm	Negate		Rd=-Rm			N Z C V	R0-R7
TST Rn,Rm	Test Masked (AND)		Flags= Rn AND Rm			N Z - -	R0-R7
B label	Branch to label					- - - -	Label= -2048 to +2048
BEQ label	Branch if Equal		Z=1			- - - -	-252 to +258
BNE label	Branch if Not Equal		Z=0			- - - -	-252 to +258
BCS label	Branch Carry Set		C=1			- - - -	-252 to +258
BHS label	Branch if Higher or Same (Unsigned)		C=1			- - - -	-252 to +258
BCC label	Branch if Carry Clear		C=0			- - - -	-252 to +258
BLO label	Branch if Lower or Same (Unsigned)		C=0			- - - -	-252 to +258
BMI label	Branch if Minus		N=1			- - - -	-252 to +258
BPL label	Branch if Plus		N=0			- - - -	-252 to +258
BVS label	Branch if oVerflow Set		V=1			- - - -	-252 to +258
BVC label	Branch if oVerflow Clear		V=0			- - - -	-252 to +258
BHI label	Branch if Higher (Unsigned)		C=1 and Z=0			- - - -	-252 to +258
BLS label	Branch if Lower or Same (Unsigned)		C=0 or Z=1			- - - -	-252 to +258
BGE label	Branch if Greater or Equal (Signed)		N=V			- - - -	-252 to +258
BLT label	Branch if Less than (Signed)		N<>V			- - - -	-252 to +258
BGT label	Branch if Greater than (Signed)		Z=0 N=V			- - - -	-252 to +258
BLE label	Branch if Less than or Equal (Signed)		Z=1 N<>V			- - - -	-252 to +258
BL label	Branch and Link		PC=label R14/LR=Return Address			- - - -	-4mb to +4mb
BX Rm	Branch and Exchange to Rm		PC=Rm			T=Bit0	R0-R15, -4mb to +4mb
SWI #	Software Interrupt					- - - -	#=0 to 255
BKPT #	Breakpoint (enter debug mode)					- - - -	#=0 to 255
ADR Rn,addr	Load address into Rn		ADD Rn,PC,#				#=0 to 1024
NOP	No operation		MOV R8,R8				
LDR Rd,[Rn,Rm]	LoaD Register (32 bit)	LDR r3,[r5,r0]				- - - -	R0-R7,#= 0 to 124 (Multiples of 4)
LDRB Rd,[Rn,Rm]	LoaD Register (8 bit)	LDRB r3,[r5,r0]				- - - -	R0-R7,#= 0 to 31 (Multiples of 1)
LDSB Rd,[Rn,Rm]	LoaD Register (Signed 8 bit)	LDRSB r3,[r5,r0]				- - - -	R0-R7,#= 0 to 31 (Multiples of 1)
LDRH Rd,[Rn,Rm]	LoaD Register (16 bit)	LDRH r3,[r5,r0]				- - - -	R0-R7,#= 0 to 62 (Multiples of 2)
LDSH Rd,[Rn,Rm]	LoaD Register (Signed 16 bit)	LDRSH r3,[r5,r0]				- - - -	R0-R7,#= 0 to 62 (Multiples of 2)
STR Rd,[Rn,Rm]	Store Register (32 Bit)	STR r3,[r5,r0]				- - - -	R0-R7,#= 0 to 124 (Multiples of 4)
STRB Rd,[Rn,Rm]	Store Register (8 Bit)	STRB r3,[r5,r0]				- - - -	R0-R7,#= 0 to 31 (Multiples of 1)
STRH Rd,[Rn,Rm]	Store Register (16 Bit)	STRH r3,[r5,r0]				- - - -	R0-R7,#= 0 to 62 (Multiples of 2)
LDR Rd,[PC,#]	LoaD Register PC relative (32 bit	LDR r3,[pc,#4]				- - - -	R0-R7,#= 0 to 124 (Multiples of 4)
LDR Rd,[SP,#]	LoaD Register SP relative (32 bit	LDR r3,[sp,#4]				- - - -	R0-R7,#= 0 to 124 (Multiples of 4)
STR Rd,[SP,#]	Store Register SP Relative (32 B	STR r3,[sp,#4]				- - - -	R0-R7,#= 0 to 124 (Multiples of 4)

ARM Complete

Command	Detail	Example	OP	Cycles	Opcode	Arch
ADCccS R0, R1, *OP2*	Add with Carry	ADC R0,R1,R2	R0 = R1+R2+C	1	0101	
ADDccS R0, R1, *OP2*	Add	ADD R0,R1,R2	R0 = R1+R2	1	0100	
ANDccS R0, R1, *OP2*	Bitwise AND	AND R0,R1,R2	R0 = R1 and R2	1	0000	
Bcc addr	Branch (JP)	B label	R15=addr	3		
BICccS R0, R1, *OP2*	Bit Clear		R0 = R1 and (CPL R2)	1	1110	
BLcc addr	Branch and Link (CALL)	BL label	R14=R15… R15=addr	3		
CDPcc #,e,Crd,Crn,Crm,e2 CDO	Return From Exception					2,5
CMNccP R1, *OP2*	Compare Negative (Set flags like ADD)		flags=R1+R2	1	1001	
CMPccP R1, *OP2*	Compare (Set flags like SUB)		flags=R1-R2	1	1010	
EORccS R0, R1, *OP2*	Exclusive OR (XOR)		R0 = R1 xor R2	1	0001	
LDCccLTN #,Crd,addr,L LDC2	Load Coprocessor					2,5
LDMccmm R0!,{R1,R2...R3}	Load Multiple (POP)		Move R1,R2...R3→(R0)	1+		
LDRccBT R0,addr,*shft*	LoaD Register (B=8 bit / T=access in user mode)		R0=(addr)	3+	psuedo	
LDRccH R0,addr,*shft*	LoaD Register (16 bit)		R0=(addr)	3+		4T+
LDRccSB R0,addr,*shft*	LoaD Register (8 bit signed)		R0=(addr)	3+		4
LDRccSH R0,addr,*shft*	LoaD Register (16 bit signed)		R0=(addr)	3+		4
MCRcc #,e,Rd,Crn,Crm,e2 MCR2	Move from registers to coprocessor					2,5,5Ed
MLAccS R0,R1,R2,R3	Multiply with Accumulate		R0=(R1*R2)+R3	16		2
MOVccS R0, *R2 ,shft*	Move		R0 = R2	1	1101	
MRCcc #,e,Rd,Crn,Crm,e2 ,MRC2	Coprocessor Register transfer					2,5
MRScc R0,flags	Move from CPSR/SPSR to register… MSR R0,CPSR	MRS R4, CPSR	=PSR			3
MSRcc fields,#n/R0	Move from register to CPSR	MSR CPSR, R4	PSR=Rm			3
MULccS R0, R1, R2	Multiply		R0=R1*R2	16		2
MVNccS R0, R2 *,shft*	Move Not (Flip bits of R2)		R0 = -R2	1	1111	
ORRccS R0, R1, R2 *,shft*	Inclusive Or		R0 = R1 or R2	1	1100	
RSBccS R0, R1, R2 *,shft*	Reverse SuBtract		R0 = R2-R1	1	0011	
RSCccS R0, R1, R2 *,shft*	Reverse Subtract with Carry		R0 = R2-R1+C-1	1	0111	
SBCccS R0, R1, R2 *,shft*	Subtract with carry		R0 = R1-R2+C-1	1	0110	
STCccLTN #,Crd,addr,L STC2	Store to Coprosessor					2,5ExP
STMccmm R0,{R1,R2...R3}!	Store Multiple (PUSH)		Restore (R0)-> R1,R2...	2+		
STRccBT R0,(addr),*shft*	Store Register (32 Bit)		(addr)=R0	2+		
STRccH R0,(addr)	Store Register (16 bit)		(addr)=R0	2+ (H=4+)		4T+
SUBccS R0, R1, R2 *,shft*	Subtract		R0 = R1-R2	1	0010	
SWIcc #n	Software Interrupt (RST)			3		
SWPccB r0,r1,[base]	Load r0 from [base],store r1 in [base]		Rd=Rn… Rn=Rd			3
TEQccP R1, R2 *,shft*	Test Inverted (EOR) (P=set flags)	teqp R4,#0	flags=R1 xor R2	1	1001	
TSTccP R1, R2 *,shft*	Test Masked (AND) (P=set flags)		flags=R1 AND R2	1	1000	
ADRcc Rn,addr	Load relative address into register		R0=addr		psuedo	
ADRccL Rn,label	Load Long relative address into register				psuedo	
NOP	no operation				psuedo	
P - Alter Processor Flags	S – Set condition codes	cc – Condition Code		B – Byte		
H - 16 Bit D - 64 bit	T- Translation (User Privilages in Super mode)					

Arm 5+

Command	Detail	Example	OP	Cycles	Opcode	Arch
BXJcc R0	Branch and change to Jazelle state					6
BKPT imm	Breakpoint					5
BLX addr,BLXcc R0	Branch , link and exchange					5Tb
BXcc R0	Branch and exchange		R15=Rn… Tbit=Rn[0]			5tb
CLZcc R0, R1	Count Leading Zeros					5
CPSeeff #n	Change Processor state					6
CPYcc R0, R1	Copy one register to another		R0=R1			6
LDRccD R0,addr	LoaD Register (64 bit)		R0=(addr),R1=(addr+4)	3+		5TE
LDREXcc R0,R1	LoaD Register and set memory exclusive		R0=(R1)	3+		6
MAR	Mover from registers to 40 bit acc					Xscale
MCRRcc #,e,Rd,Rn,Crn,Crm,e2 M	Move from 2 registers to coprocessor					5TE,6
MIA,MIAPH,MIAxy	Multiply with internal 40 bit accumulate					Xscale
MRA	Multiply from 40 bit accumulator to registers					Xscale
MRRCcc #,e,Rd,Rn,Crm, MRC2	Move from coprocessor to 2 regs					5E
PKHBTcc R0, R1, R2 *,shft*	Pack Halfword Bottom/Top (L from R1 / H from R2)		R0=R2H+R1L			6
PKHTBcc R0, R1, R2 *,shft*	Pack Halfword Top/Bottom (H from R1 / L from R2)		R0=R1H+R2L			6
PLD mode	Cache Preload					5E
QADDcc R0, R1, R2	Saturating Arithmatic					5Exp
QADD16cc R0, R1, R2	Saturating Arithmatic (16 bit)					6
QADD8cc R0, R1, R2	Saturating Arithmatic (8 bit)					6
QADDSUBXcc R0, R1, R2	Saturating Add and Subtract with Exchange					6
QDADDcc R0, R1, R2	Saturating Double and Add					5TE
QDSUBcc R0, R1, R2	Saturating Double and Subtract					5TE
QSUBcc R0, R1, R2	Saturating Subtract					5TE
QSUB16cc R0, R1, R2	Saturating Subtract (16 bit)					6
QSUB8cc R0, R1, R2	Saturating Subtract (8 bit)					6
QSUBADDXcc R0, R1, R2	Saturating Add and Subtract with Exchange					6
REVcc R0, R1	reverses the byte order in a 32-bit register.					6
REV16cc R0, R1	reverses the byte order in a 16-bit register.					6
REVSHcc R0, R1	reverses the byte order in a 16-bit register, and sign extend					6
RFE<mode> R0!	Return From Exception					6
SADD16cc R0, R1, R2	Signed Add two 16 bit numbers					6
SADD8cc R0, R1, R2	Signed Add four 8-bit signed integer additions					6
SADDSUBXcc R0, R1, R2	Signed 16-bit Add and Subtract with Exchange					6
SELcc R0, R1, R2	Select bytes from R1/R2 based on GE flags					6

SETEND <endian>	Set Endian mode					6
SHADD16cc R0, R1, R2	Signed Halving Add (16 bit)					6
SHADD8cc R0, R1, R2	Signed Halving Add (8 bit)					6
SHADDSUBXcc R0, R1, R2	Signed Halving Add and Subtract with Exchange (16 bit)					6
SHSUB16cc R0, R1, R2	Signed Halving Subtract (16 bit)					6
SHSUB8cc R0, R1, R2	Signed Halving Subtract (8 bit)					6
SHSUBADDXcc R0, R1, R2	Signed Halving Subtract and Add with Exchange (16 bit)					6
SMLALxycc R0L, R1H, R2,R3	Signed Multiply-accumulate Long					5TE
SMLAxycc	Signed Multiply-accumulate					5TE
SMLADXcc	Signed Multiply-accumulate Dual					6
SMLALccS R0L, R1H, R2,R3	Signed Multiply-accumulate Long					6
SMLAWycc	Signed Multiply-accumulate Word B and T					5ExP
SMLSDXcc R0, R1, R2,R3	Signed Multiply Subtract accumulate Dual					6
SMLSLDXcc R0, R1, R2,R3	Signed Multiply Subtract accumulate LongDual					6
SMMLARcc R0, R1, R2,R3	Signed Most significant word Multiply Accumulate					6
SMMLSRcc R0, R1, R2,R3	Signed Most significant word Multiply Subtract					6
SMULLRcc R0, R1, R2	Signed Multiply (R=Round)					6
SMUADXcc R0, R1, R2	Signed Dual Multiply Add					6
SMULXYcc R0, R1, R2	Signed Multiply BB , BT , TB , or TT					ARMv5TE
SMULLcc R0L, R1H, R2,R3	Signed Multiply Long					ARMv5TE
SMULWYcc R0, R1, R2	Signed Multiply Word B and T					ARMv5TE
SMUSDXcc R0, R1, R2	Signed Dual Multiply Subtract					6
SRS<Mode> #mode**!**	Store Return State					6
SSAT16cc R0,#n, R1,*shft*	Signed Saturate (16 bit)					6
SSATcc R0,#n, R1,*shft*	Signed Saturate					6
SSUB16cc R0, R1, R2	Signed Subtract (16 bit)					6
SSUB8cc R0, R1, R2	Signed Subtract (8 bit)					6
SSUBADDXcc R0, R1, R2	Signed Subtract and Add with Exchange (16 bit)					6
STRccD R0,(addr)	Store Register (64 bit)		(addr)=R0,(addr+4)=R1	2+		ARMv5TE
STREXcc R0,R1,R2	Store Register Exclusive					6
SXTABcc R0,R1,R2,*shft*	Extract an 8 bit value, and sign extend					6
SXTAB16cc R0,R1,R2,*shft*	Extract two 8 bit value, and sign extend to 16 bits					6
SXTAHcc R0,R1,R2,*shft*	Extract a 16 bit value, and sign extend					6
SXTBcc R0,R1,*shft*	Take a 8-bit value from a register and sign extends it to 32 bits.					6
SXTB16cc R0,R1,*shft*	Take two 8-bit value from a register and sign extends it to 16 bits.					6
SXTHcc R0,R1,*shft*	Take two 16-bit value from a register and sign extend to 32 bits					6
UADD16cc R0,R1,R2	Unsigned Add (16 bit)					6
UADD8cc R0,R1,R2	Unsigned Add (8 bit)					6
UADDSUBXcc R0,R1,R2	Unsigned Add and Subtract with Exchange					6
UHADD16cc R0,R1,R2	Unsigned Halving Add (16 bit)					6
UHADD8cc R0,R1,R2	Unsigned Halving Add (8 bit)					6
UHSUB16cc R0,R1,R2	Unsigned Halving Subtract (16 bit)					6
UHSUB8cc R0,R1,R2	Unsigned Halving Subtract (8 bit)					6
USUBADDXcc R0,R1,R2	Unsigned Subtract and Add with Exchange					6
UMAALccS R0L, R1H, R2,R3	Unsigned Multiply Accumulate Long					6
UMULLccS R0L, R1H, R2,R3	Unsigned Multiply Long					6
UQADD16cc R0,R1,R2	Unsigned Saturating Add (16 bit)					6
UQADD8cc R0,R1,R2	Unsigned Saturating Add (8 bit)					6
UQADDSUBXcc R0,R1,R2	Unsigned Saturating Add and Subtract with Exchange					6
UQSUB16cc R0,R1,R2	Unsigned Saturating Subtract (16 bit)					6
UQSUB8cc R0,R1,R2	Unsigned Saturating Subtract (8 bit)					6
UQSUBADDXcc R0,R1,R2	Unsigned Saturating Subtract and Add with Exchange					6
USAD8cc R0,R1,R2	Unsigned Sum of Absolute Differences					6
USADA8cc R0,R1,R2,R3	Unsigned Sum of Absolute Differences and Accumulate					6
USATcc R0,#n, R1,*shft*	Unsigned Saturate					6
USAT16cc R0,#n, R1,*shft*	Unsigned Saturate (16 bit)					6
USUB16cc R0,R1,R2	Unsigned Subtract (16 bit)					6
USUB8cc R0,R1,R2	Unsigned Subtract (8 bit)					6
USUBADDXcc R0,R1,R2	Unsigned Subtract and Add with Exchange					6
UXTABcc R0,R1,R2,*shft*	Extract an 8 bit value and Zero extend					6
UXTAB16cc R0,R1,R2,*shft*	Extract two 8 bit values and Zero extend					6
UXTAHcc R0,R1,R2,*shft*	Extract an 16 bit value and Zero extend					6
UXTBcc R0,R1,*shft*	Extract an 8 bit value and Zero extend					6
UXTB16cc R0,R1,*shft*	Extract two 8 bit values and Zero extend					6
UXTHcc R0,R1,*shft*	Extract a 16 bit value and Zero Extend					6
P - Alter Processor Flags	**S – Set condition codes**	**cc – Condition Code**		**B – Byte**		
H - 16 Bit **D** - 64 bit	**T- Translation (User Privilages in Super mode)**					

Power PC

Opcode	Integer Arithmetic Instructions	Details
addi rd,ra,simm	Add Immediate	The sum (rAID) + SIMM is placed into register rD.
addis rd,ra,simm	Add Immediate Shifted	The sum (rAID) + (SIMM II x '0000') is placed into register rD.
add rD,rA,rB	Add	The sum (rA) + (rB) is placed into register rD.
add. rD,rA,rB	Add with CR Update.	The dot suffix enables the update of the condition register.
addo rD,rA,rB	Add with Overflow Enabled.	The o suffix enables the overflow bit (OV) in the XER.
addo. rD,rA,rB	Add with Overflow and CR Update.	The o. suffix enables the update of the condition register and enables the overflow bit (OV) in the XER.
subf rD,rA,rB	Subtract from	The sum --. (rA) + (rB) + 1 is placed into rD
subf. rD,rA,rB	Subtract from with CR Update.	The dot suffix enables the update of the condition register.
subfo rD,rA,rB	Subtract from with Overflow Enabled.	The o suffix enables the overflow. The o suffix enables the overflow bit (OV) in the XER.
subfo. rD,rA,rB	Subtract from with Overflow and CR Update	The o. suffix enables the update of the condition register and enables the overflow bit (OV) in the XER.
addic rD,rA,SIMM	Add Immediate Carrying	The sum (rA) + SIMM is placed into register rD.
addic. rD,rA,SIMM	Add Immediate Carrying and Record	The sum (rA) + SIMM is placed into rD. The condition register is Immediate updated.
subflc rD,rA,SIMM	Subtract from Immediate Carrying	The sum --. (rA) + SIMM + 1 is placed into register rD.
addc rD,rA,rB	Add Carrying	The sum (rA) + (rB) is placed into register rD.
addc. rD,rA,rB	Add Carrying with CR Update.	The dot suffix enables the update of the condition register.
addco rD,rA,rB	Add Carrying with Overflow Enabled.	The o suffix enables the overflow bit (OV) in the XER.
addco. rD,rA,rB	Add Carrying with Overflow and CR Update.	The o. suffix enables the update of the condition register and enables the overflow bit (OV) in the XER.
subfc rD,rA,rB	Subtract from Carrying	The sum -, (rA) + (rB) + 1 is placed into register rD.
subfc. rD,rA,rB	Subtract from Carrying with CR Update.	The dot suffix enables the update of the condition register.
subfco rD,rA,rB	Subtract from Carrying with Overflow.	The o suffix enables the overflow bit (OV) in the XER.
subfco. rD,rA,rB	Subtract from Carrying with Overflow and CR Update.	The o. suffix enables the update of the condition register and enables the overflow bit (OV) in the XER.
adde rD,rA,rB	Add Extended	The sum (rA) + (rB) + XER(CA) is placed into register rD.
adde. rD,rA,rB	Add Extended with CR Update.	The dot suffix enables the update of the condition register.
addeo rD,rA,rB	Add Extended with Overflow.	The o suffix enables the overflow bit (OV) in the XER.
addeo. rD,rA,rB	Add Extended with Overflow and CR Update.	The o. suffix enables the update of the condition register and enables the overflow bit (OV) in the XER.
subfe rD,rA,rB	Subtract from Extended	The sum -,(rA) + (rB) + XER(CA) is placed into register rD.
subfe. rD,rA,rB	Subtract from Extended with CR Update.	The dot suffix enables the update of the condition register.
subfeo rD,rA,rB	Subtract from Extended with Overflow.	The o suffix enables the overflow bit (OV) in the XER.
subfeo. rD,rA,rB	Subtract from Extended with Overflow and CR Update.	The o. suffix enables the update of the condition register and enables the overflow (OV) bit in the XER.
addme rD,rA	Add to Minus One Extended	The sum (rA) + XER(CA) + x'FFFFFFFF' is placed into register rD.
addme. rD,rA	Add to Minus One Extended with CR Update.	The dot suffix enables the update of the condition register.
addmeo rD,rA	Add to Minus One Extended with Overflow.	The o suffix enables the overflow bit (OV) in the XER.
addmeo. rD,rA	Add to Minus One Extended with Overflow and CR Update.	The o. suffix enables the update of the condition register and enables the overflow (OV) bit in the XER.
subfme rD,rA	Subtract from Minus One Extended	The sum..., (rA) + XER(CA) + x'FFFFFFFF' is placed into register rD.
subfme. rD,rA	Subtract from Minus One Extended with CR Update.	The dot suffix enables the update of the condition register.
subfmeo rD,rA	Subtract from Minus One Extended with Overflow.	The o suffix enables the overflow bit (OV) in the XER.
subfmeo. rD,rA	Subtract from Minus One Extended with Overflw & CR updt.	The o. suffix enables the update of the condition register and enables the overflow bit (OV) in the XER.
addze rD,rA	Add to Zero Extended	The sum (rA) + XER(CA) is placed into register rD.
addze. rD,rA	Add to Zero Extended with CR Update.	The dot suffix enables the update of the condition register.
addzeo rD,rA	Add to Zero Extended with Overflow.	The o suffix enables the overflow bit (OV) in the XER.
addzeo. rD,rA	Add to Zero Extended with Overflow and CR Update.	The o. suffix enables the update of the condition register and enables the overflow bit (OV) in the XER.
subfze rD,rA	Subtract from Zero Extended	The sum..., (rA) + XER(CA) is placed into register rD.
subfze. rD,rA	Subtract from Zero Extended with CR Update.	The dot suffix enables the update of the condition register.
subfzeo rD,rA	Subtract from Zero Extended with Overflow.	The 0 suffix enables the overflow bit (OV) in the XER.
subfzeo. rD,rA	Subtract from Zero Extended with Overflow and CR Update.	The o. suffix enables the update of the condition register and enables the overflow bit (OV) in the XER.
neg rD,rA	Negate	The sum..., (rA) + 1 is placed into register rD.
neg. rD,rA	Negate with CR Update.	The dot suffix enables the update of the condition register.
nego rD,rA	Negate with Overflow.	The o suffix enables the overflow bit (OV) in the XER.
nego. rD,rA	Negate with Overflow and CR Update.	The o. suffix enables the update of the condition register and enables the overflow bit (OV) in the XER.
mulli rD,rA,SIMM	Multiply Low Immediate	The low-order 32 bits of the 48-bit product (rA)*SIMM are placed into register rD. The low-order 32 bits of the product are the correct 32-bit product. The low-order bits are independent of whether the operands are treated as signed or unsigned integers. However, XER[OV] is set based on the result interpreted as a signed integer. The high-order bits are lost. This instruction can be used with mulhwxto calculate a full 64-bit product.
mullw rD,rA,rB	Multiply Low	The low-order 32 bits of the 64-bit product (rA) *(rB) are placed into Low register rD. The low-order 32 bits of the product are the correct 32-bit product. The low-order bits are independent of whether the operands are treated as signed or unsigned integers. However, XER[OV] is set based on the result interpreted as a signed integer. The high-order bits are lost. This instruction can be used with mulhwxto calculate a full 64-bit product. Some implementations may execute faster if rB contains the operand having the smaller absolute value.
mullw. rD,rA,rB	Multiply Low with CR Update.	The dot suffix enables the update of the condition register.
mullwo rD,rA,rB	Multiply Low with Overflow.	The o suffix enables the overflow bit (OV) in the XER.
mullwo. rD,rA,rB	Multiply Low with Overflow and CR Update.	The o. suffix enables the update of the condition register and enables the overflow bit (OV) in the XER.
mulhw rD,rA,rB	Multiply High Word	The contents of rA and rB are interpreted as 32-bit signed integers.The 64-bit product is formed. The high-order 32 bits of the 64-bit product are placed into Rd. Both operands and the product are interpreted as signed integers. This instruction may execute faster if rB contains the operand having the smaller absolute value.
mulhw. rD,rA,rB	Multiply High Word with CR Update.	The dot suffix enables the update of the condition register.
mulhwu rD,rA,rB	Multiply High Word Unsigned	The contents of rA and of rB are extracted and interpreted as 32-bit unsigned integers. The 64-bit product is formed. The high-order 32 Unsigned bits of the 64-bit product are placed into rD. Both operands and the product are interpreted as unsigned integers. This instruction may execute faster if rB contains the operand having the smaller absolute value.
mulhwu. rD,rA,rB	Multiply High Word Unsigned with CR Update.	The dot suffix enables the update of the condition register.
divw rD,rA,rB	Divide Word	The dividend is the signed value of (rA). The divisor is the signed value of (rB). The 64-bit quotient is formed. The low-order 32 bits of the 64-bit quotient are placed into rD. The remainder is not supplied as a result.Both operands are interpreted as signed integers. The quotient is the unique signed integer that satisfies the following: dividend = (quotient times divisor) + r where 0 <= r < Idivisorl if the dividend is non-negative, and -Idivisorl <r <= 0 if the dividend is negative. If an attempt is made to perform any of the divisions x'8000_0000' /-1 or <anything> / 0 the contents of register rD are undefined, as are the contents of the LT, GT, and EQ bits of the condition register field CRO if the instruction has condition register updating enabled. In these cases, if instruction overflow is enabled, then XER[OV] is set. The 32-bit signed remainder of dividing (rA) by (rB) can be computed as follows, except in the case that (rA) = _2 31 and (rB) = -1: divw rD,rA,rB rD = quotient mull rD,rD,rB rD = quotient*divisor subf rD,rD,rA rD = remainder
divw. rD,rA,rB	Divide Word with CR Update.	The dot suffix enables the update of the condition register.
divwo rD,rA,rB	Divide Word with Overflow.	The o suffix enables the overflow bit (OV) in the XER.
divwo. rD,rA,rB	Divide Word with Overflow and CR Update.	The o. suffix enables the update of the condition register and enables the overflow bit (OV) in the XER.
divwu rD,rA,r8	Divide Word Unsigned	The dividend is the value of (rA). The divisor is the value of (r8). The 32 bit quotient is placed into rD. The remainder is not supplied as a result.Both operands are interpreted as unsigned integers. The quotient is the unique unsigned integer that satisfies the following: dividend = (quotient times divisor) + r where 0 <= r < divisor. If an attempt is made to perform the division <anything> / 0 the contents of register rD are undefined, as are the contents of the LT, GT, and EQ bits of the condition register field CRO if the instruction has the condition register updating enabled. In these cases, if instruction overflow is enabled, then XER[OV] is set. The 32-bit unsigned remainder of dividing (rA) by (r8) can be computed as follows: divwu rD,rA,r8 rD = quotient mull rD,rD,r8 rD = quotient*divisor subf rD,rD,rA rD = remainder
divwu. rD,rA,rB	Divide Word Unsigned with CR Update.	The dot suffix enables the update of the condition register.
divwuo rD,rA,rB	Divide Word Unsigned with Overflow.	The a suffix enables the overflow bit (OV) in the XER.
divwuo. rD,rA,rB	Divide Word Unsigned with Overflow and CR Update.	The o. suffix enables the update of the condition register and enables the overflow bit (OV) in the XER.

Opcode	Integer Compare Instructions	Details
cmpi crfD,L,rA,SIMM	Compare Immediate	The contents of register rA is compared with the sign-extended value of the SIMM operand, treating the operands as signed integers. The result of the comparison is placed into the CR field specified by operand crfD.
cmp crfD,L,rA,rB	Compare	The contents of register rA is compared with register rB, treating the operands as signed integers. The result of the comparison is placed into the CR field specified by operand crfD.
cmpli crfD,L,rA,UIMM	Compare Logical Immediate	The contents of register rA is compared with x'OOOO' II UIMM, treating the operands as unsigned integers. The result of the comparison is placed into the CR field specified by operand crfD.
cmpl crfD,L,rA,rB	Compare Logical	The contents of register rA is compared with register rB, treating the operands as unsigned integers. The result of the comparison is placed into the CR field specified by operand crfD.
cmpwi crfD,rA,SIMM	Compare Word Immediate	Equivalent to: cmpi crfD,O,rA,SIMM
cmpw crfD,rA,rB	Compare Word	Equivalent to: cmp crfD,O,rA,rB
cmplwi crfD,rA,UIMM	Compare Logical Word Immediate	Equivalent to: cmpli crfD,O,rA,UIMM
cmplw crfD,rA,rB	Compare Logical Word	Equivalent to: cmpl crfD,O,rA,rB

Opcode	Integer Logical Instructions	Details
andi. rA,rS,UIMM	AND Immediate	The contents of rS is ANDed with x'OOOO' II UIMM and the result is placed into rA.
andis. rA,rS,UIMM	AND Immediate Shifted	The contents of rS is ANDed with UIMM II x'OOOO' and the result is placed into rA.
ori rA,rS,UIMM	OR Immediate	The contents of rS is ORed with x'OOOO' II UIMM and the result is placed into rA. The preferred no-op is ori 0,0,0
oris rA,rS,UIMM	OR Immediate Shifted	The contents of rS is ORed with UIMM IIx'OOOO' and the result is placed into rA.
xori rA,rS,UIMM	XOR Immediate	The contents of rS is XORed with x'OOOO' II UIMM and the result is placed into rA.
xoris rA,rS,UIMM	XOR Shifted	The contents of rS is XORed with UIMM IIx'OOOO' and the result is Immediate placed into rA.
and rA,rS,rB	AND	The contents of rS is ANDed with the contents of register rB and the result is placed into rA.
and. rA,rS,rB	AND with CR Update.	The dot suffix enables the update of the condition register.
or rA,rS,rB	OR	The contents of rS is ORed with the contents of rB and the result is placed into rA.
or. rA,rS,rB	OR with CR Update.	The dot suffix enables the update of the condition register.
xor rA,rS,rB	XOR	The contents of rS is XORed with the contents of rB and the result is placed into register rA.
xor. rA,rS,rB	XOR with CR Update.	The dot suffix enables the update of the condition register.
nand rA,rS,rB	NAND	The contents of rS is ANDed with the contents of rB and the one's complement of the result is placed into register rA. NAND with rA=rB can be used to obtain the one's complement.
nand. rA,rS,rB	NOR with CR Update.	The dot suffix enables the update of the condition register.
eqv rA,rS,rB	Equivalent	The contents of rS is XORed with the contents of rB and the complemented result is placed into register rA.
eqv. rA,rS,rB	Equivalent with CR Update.	The dot suffix enables the update of the condition register.
andc rA,rS,rB	AND with Complement	The contents of rS is ANDed with the complement of the contents of andc. rB and the result is placed into rA.
andc. rA,rS,rB	AND with Complement with CR Update.	The dot suffix enables the update of the condition register.

orc rA,rS,rB	OR with Complement	The contents of rS is ORed with the complement of the contents of rB and the result is placed into rA.
orc. rA,rS,rB	OR with Complement with CR Update.	The dot suffix enables the update of the condition register.
extsb rA,rS	Extend Sign Byte	Register r S[24-31] are placed into rA[24-31]. Bit 24 of rS is placed into rA[O-23].
extsb. rA,rS	Extend Sign Byte with CR Update.	The dot suffix enables the update of the condition register.
extsh rA,rS	Extend Sign Half Word	Register r S[16-31] are placed into rA[16-31]. Bit 16 of rS is placed into rA[O-15].
extsh. rA,rS	Extend Sign Half Word with CR Update.	The dot suffix enables the update of the condition register.
cntlzw rA,rS	Count Leading Zeros Word	A count of the number of consecutive zero bits of rS is placed into rA. This number ranges from 0 to 32, inclusive.
cntlzw. rA,rS	Count Leading Zeros Word with CR Update.	The dot suffix enables the update of the condition register. When the Count Leading Zeros Word instruction has condition register updating enabled, the LT field is cleared to zero in CR0.
Opcode	**Integer Rotate Instructions**	**Details**
rlwinm rA,rS,SH,MB,ME	Rotate Left Word Immediate then AND with Mask	The contents of register rS are rotated left by the number of bits specified by operand SH. A mask is generated having 1-bits from the bit specified by operand MB through the bit specified by operand ME and a-bits elsewhere. The rotated data is ANDed with the generated mask and the result is placed into register rA. Simplified mnemonics: extlwi rA,rS,n,b rlwinm rA,rS,b,O,n-1 srwi rA,rS,n rlwinm rA,rS,32-n,n,31 clrrwi rA,rS,n rlwinm rA,rS,O,O,31-n Note: The rlwinm instruction can be used for extracting, clearing and shifting bit fields using the methods shown below: To extract an n-bit field that starts at bit position b in register rS, , right-justified into rA (clearing the remaining 32-n bits of rA), set SH",b+n, MB",32-n, and ME",31. To extract an n-bit field that starts at bit position bin rS, left-justified into rA, set SH=b, MB '" a, and ME",n-1. To rotate the contents of a register left (right) by n bits, set SH",n (32-n), MB=O, and ME",31. To shift the contents of a register right by n bits, set SH=32-n, MB",n, and ME",31. To clear the high-order b bits of a register and then shift the result left by n bits, set SH",n, MB",b-n and ME",31-n. To clear the low-order n bits of a register, set SH=O, MB",O, and ME=31-n.
rlwinm. rA,rS,SH,MB,ME	Rotate Left Word Imm & AND with Mask with CR Update.	The dot suffix enables the update of the condition register.
rlwnm rA,rS,rB,MB,ME	Rotate Left Word then AND with Mask	The contents of rS are rotated left by the number of bits specified by rB[27-31]. A mask is generated having 1-bits from the bit specified by operand MB through the bit specified by operand ME and O-bits elsewhere. The rotated data is ANDed with the generated mask and the result is placed into rA.rotlw rA,rS,rB rlwnm rA,rS,rB,O,31 Note: The rlwinm instruction can be used to extract and rotate bit fields using the methods shown below: To extract an n-bit field that starts at the variable bit position bin the register specified by operand rS, right-justified into rA (clearing the remaining 32-nbits of rA), set r B[27-31]=b+n, MB=32-n, and ME=31. To extract an n-bit field that starts at variable bit position b in the register specified by operand rS, left-justified into rA (clearing theremaining 32-n bits of rA), set rB[27-31]=b, MB = 0, and ME=n-1. To rotate the contents of the low-order 32 bits of a register left (right) by variable n bits, set rB[27-31]=n (32-n), MB=O, and ME=31.
rlwnm. rA,rS,rB,MB,ME	Rotate Left Word then AND with Mask with CR Update.	The dot suffix enables the update of the condition register.
rlwimi rA,rS,SH,MB,ME	Rotate Left Word Immediate then Mask	The contents of rS are rotated left by the number of bits specified by operand SH. A mask is generated having 1-bits from the bit specified by MB through the bit specified by ME and O-bits elsewhere. The rotated data is inserted into rA under control of the generated mask. Simplified mnemonic: inslw rA,rS,n,b rlwim rA,rS,32-b,b,b+n-1 Note: The opcode rlwimi can be used to insert a bit field into the contents of register specified by operand rA using the methods shown below: To insert an n-bit field that is left-justified in rS into rA starting at bit position b, set SH=32-b, MB=b, and ME=(b+n)-1. To insert an n-bit field that is right-justified in rS into rA starting at bit position b, set SH=32-(b+n), MB=b, and ME=(b+n)-1.
rlwimi. rA,rS,SH,MB,ME	Rotate Left Word Immediate & Mask Insert with CR Update.	The dot suffix enables the update of the condition register.
Opcode	**Integer Shift Instructions**	**Details**
slw rA,rS,rB	Shift Left Word	The contents of rS are shifted left the number of bits specified by slw. rB[26-31]. Bits shifted out of position 0 are lost. Zeros are supplied to the vacated positions on the right. The 32-bit result is placed into rA. It rB[26]=1, then rA is filled with zeros.
slw. rA,rS,rB	Shift Left Word with CR Update.	The dot suffix enables the update of the condition register.
srw rA,rS,rB	Shift Right Word	The contents of rS are shifted right the number of bits specified by Word srw. rB[26-31]. Zeros are supplied to the vacated positions on the left. The 32-bit result is placed into rA. It rB[26]=1, then rA is filled with zeros.
srw. rA,rS,rB	Shift Right Word with CR Update.	The dot suffix enables the update of the condition register.
srawi rA,rS,SH	Shift Right Algebraic Word Immediate	The contents of rS are shifted right the number of bits specified by operand SH. Bits shifted out of position 31 are lost. The 32-bit result is sign extended and placed into rA. XER[CA] is set if r S contains a negative number and any 1-bits are shifted out of position 31 ; otherwise XER(CA) is cleared. An operand SH of zero causes rA to be loaded with the contents of rS and XER[CA] to be cleared to 0.
srawi. rA,rS,SH	Shift Right Algebraic Word Immediate with CR Update.	The dot suffix enables the update of the condition register.
sraw rA,rS,rB	Shift Right Algebraic Word	The contents of rS are shifted right the number of bits specified by sraw. rB[26-31]. The 32-bit result is placed into rA. XER[CA] is set to 1 if rS contains a negative number and any 1-bits are shifted out of position 31; otherwise XER[CA] is cleared to O. An operand (rB) of zero causes rA to be loaded with the contents of rS, and XER[CA] to be cleared to O. If rB[26]= 1, then rA is filled with 32 sign bits (bit 0) from rS. If rB[26]=0, then rA is filled from the left with sign bits. Condition register field CRO is set based on the value written into rA.
sraw. rA,rS,rB	Shift Right Algebraic Word with CR Update.	The dot suffix enables the update of the condition register.
Opcode	**Integer Load Instructions**	**Details**
lbz rD,d(rA)	Load Byte and Zero	The effective address is the sum (rAI0)+d. The byte in memory addressed by the EA is loaded into register rD[24-31]. The remaining bits in register rD are cleared to 0.
lbzx rD,rA,rB	Load Byte and Zero Indexed	The effective address is the sum (rAI0)+(rB). The byte in memory addressed by the EA is loaded into register rD[24-31]. The remaining bits in register rD are cleared to 0.
lbzu rD,d(rA)	Load Byte and Zero with Update	The effective address (EA) is the sum (rAI0)+d. The byte in memory addressed by the EA is loaded into register rD[24-31]. The remaining bits in register rD are cleared to 0. The EA is placed into register rA. If operand rA=0 the MPC601 does not update rD, or if rA=rD the load data is loaded into register rD and the register update is suppressed. Although the PowerPC architecture defines load with update instructions with operand rA=0 or rA=rD as invalid forms, the MPC601 allows these cases.
lbzux rD,rA,rB	Load Byte and Zero with Update Indexed	The effective address (EA)is the sum (rAI0)+(rB). The byte addressed by the EA is loaded into register rD[24-31]. The remaining bits in register rD are cleared to 0. The EA is placed into register rA. If operand rA=O the MPC601 does not update register rD, or if rA=rD the load data i5 loaded into register rD and the register update is suppressed. Although the PowerPC architecture defines load with update instructions with operand rA=0 or rA=rD as invalid forms, the MPC601 allows these cases.
lhz rD,d(rA)	Load Half Word and Zero	The effective address is the sum (rAI0)+d. The half-word in memory addressed by the EA is loaded into register rD[16-31]. The remaining bits in rD are cleared to 0.
lhzx rD,rA,rB	Load Half Word and Zero Indexed	The effective address is the sum (rAI0)+(rB). The half-word in memory addressed by the EA is loaded into register rD[16-31]. The remaining bits in register rD are cleared.
lhzu rD,d(rA)	Load Half Word and Zero with Update	The effective address is the sum (rAIO)+d. The half-word in memory addressed by the EA is loaded into register rO[l6-31]. The remaining bits in register rO are cleared. The EA is placed into register rA. If operand rA=O the MPC601 does not update register rO, or if rA=rO the load data is loaded into register rO and the register update is suppressed. Although the PowerPC architecture defines load with update instructions with operand rA=O or rA=rO as invalid forms, the MPC601 allows these cases.
lhzux rD,rA,rB	Load Half Word and Zero with Update Indexed	The effective address is the sum (rAI0)+(rB). The half-word in memory addressed by the EA is loaded Into register rD[l6-31]. The remaining bits in register rD are cleared. The EA is placed into register rA. Although the PowerPC architecture defines load with update instructions with operand rA=0 or rA=rD as invalid forms, the MPC601 allows these cases.
lha rD,d(rA)	Load Half Word Algebraic	The effective address is the sum (rA)+d. The half-word in memory addressed by the EA Is loaded into register rD[l6-31]. The remaining bits In register rD are filled with a copy of bit 0 of the loaded half-word.
lhax rD,rA,rB	Load Half Word Algebraic Indexed	The effective address is the sum (rAIO)+(rB). The half-word in memory addressed by the EA is loaded into register rO[16-31]. The remaining bits in register rD are filled with a copy of bit 0 of the loaded half-word.
lhau rD,d(rA)	Load Half Word Algebraic with Update	The effective address is the sum (rAI0)+d. The half-word in memory addressed by the EA is loaded into register rD[16-31]. The remaining bits in register rO are filled with a copy of bit 0 of the loaded half-word. The EA is placed into register rA. If operand rA=0 the MPC601 does not update register rD, or if rA=rD the load data is loaded into register rD and the register update is suppressed. Although the PowerPC architecture defines load with update instructions with operand rA=0 or rA=rD as invalid forms, the MPC601 allows these cases.
lhaux rD,rA,rB	Load Half Word Algebraic with Update Indexed	The effective address Is the sum (rAI0)+(rB). The half-word In memory addressed by the EA is loaded into register rD[l6-31]. The remaining bits in register rD are filled with a copy of bit 0 of the loaded half-word. The EA is placed into register rA. If operand rA=0 the MPC601 does not update rD, or if rA=rD the load data is loaded into register rD and the register update Is suppressed. Although the PowerPC architecture defines load with update instructions with operand rA=0 or rA=rD as Invalid forms, the MPC601 allows these cases.
lwz rD,d(rA)	Load Word and Zero	The effective address is the sum (rAI0)+d. The word in memory addressed by the EA is loaded into register rD[0-31].
lwzx rD,rA,rB	Load Word and Zero Indexed	The effective address is the sum (rAI0)+(rB). The word in memory addressed by the EA is loaded into register rD[0-31].
lwzu rD,d(rA)	Load Word and Zero with Update	The effective address is the sum (rAI0)+d. The word in memory addressed by the EA is loaded into register rD[0-31]. The EA is placed into register rA. If operand rA=0 the MPC601 does not update register rD, or if rA=rD the load data is loaded into register rD and the register update is suppressed. Although the PowerPC architecture defines load with update instructions with operand rA=0 or rA=rD as invalid forms, the MPC601 allows these cases.
lwzux rD,rA,rB	Load Word and Zero with Update Indexed	The effective address is the sum (rAIO)+(rB). The word in memory addressed by the EA is loaded into register rD[0-31]. The EA is placed into register rA. If operand rA=O the MPC601 does not update register rO, or if rA=rD the load data is loaded into register rD and the register update is suppressed. Although the PowerPC architecture defines load with update instructions with operand rA=O or rA=rD as invalid forms, the MPC601 allows these cases.
Opcode	**Integer Store Instructions**	**Details**
stbu rS,d(rA)	Store Byte with Update	The effective address is the sum (rAI0)+d. rS[24-31] is stored into the byte in memory addressed by the EA. The EA is placed into register rA.
stbux rS,rA,rB	Store Byte with Update Indexed	The effective address is the sum (rAI0)+(rB). rS[24-31] is stored into the byte in memory addressed by the EA. The EA is placed into register rA.
sth rS,d(rA)	Store Half word	The effective address is the sum (rAIO)+d. rS[16-31] is stored into the half-word in memory addressed by the EA.
sthx rS,rA,rB	Store half-word Indexed	The effective address (EA) is the sum (rAIO)+(rB). rS[16-31] is stored into the half-word in memory addressed by the EA.
sthu rS,d(rA)	Store Half word with Update	The effective address is the sum (rAIO)+d. rS[16-31] is stored into the half-word in memory addressed by the EA. The EA is placed into register rA.
sthux rS,rA,rB	Store Half word with Update Indexed	The effective address is the sum (rAIO)+(rB). rS[16-31] is stored into the half-word in memory addressed by the EA. The EA is placed into register rA.
stw rS,d(rA)	Store Word	The effective address is the sum (rAIO)+d. Register rS is stored into the word in memory addressed by the EA.
stwx rS,rA,rB	Store Word Indexed	The effective address is the sum (rAIO)+(rB). rS is stored into the word in memory addressed by the EA.
stwu rS,d(rA)	Store Word with Update	The effective address is the sum (rAIO)+d. Register rS is stored into the word in memory addressed by the EA. The EA is placed into register rA..
stwux rS,rA,rB	Store Word with Update Indexed	The effective address is the sum (rAIO)+(rB). Register rS is stored into the word in memory addressed by the EA. The EA is placed into register rA.
Opcode	**Integer Load and Store with Byte Reversal Instructions**	**Details**
lhbrx rD,rA,rB	Load Half Word Byte-Reverse Indexed	The effective address is the sum (rAI0)+(rB). Bits 0-7 of the half-word in memory addressed by the EA are loaded into rD[24-31]. Bits 8-15 of the half-word in memory addressed by the EA are loaded into rD[16-23]. The rest of the bits in rD are cleared to 0.
lwbrx rD,rA,rB	Load Word Byte-Reverse Indexed	The effective address is the sum (rAI0)+(rB). Bits 0-7 of the word in memory addressed by the EA are loaded into rD[24-31]. Bits 8-15 of the word in memory addressed by the EA are loaded into rO[16-23]. Bits 16-23 of the word in memory addressed by the EA are loaded into rO[8-15]. Bits 24-31 of the word in memory addressed by the EA are loaded into rD[0-7].
sthbrx rS,rA,rB	Store HallWord Byte-Reverse Indexed	The effective address is the sum (rAIO)+(rB). rS[24-31] are stored into bits 0-7 of the half-word in memory addressed by the EA. rS[16-23] are stored into bits 8-15 of the half-word in memory addressed by the EA.
stwbrx rS,rA,rB	Store Word Byte-Reverse Indexed	The effective address is the sum (rAIO)+(rB). rS[24-31] are stored into bits 0-7 of the word in memory addressed by EA. Register rS[16-23] are stored into bits 8-15 of the word in memory addressed by the EA. Register rS[8-15] are stored into bits 16-23 of the word in memory addressed by the EA. rS[0-7] are stored into bits 24-31 of the word in memory addressed by the EA.
Opcode	**Integer Load and Store Multiple Instructions**	**Details**
lmw rD,d(rA)	Load Multiple Word	The effective address is the sum (rAIO)+d. n= 32-rD. n consecutive words starting at EA are loaded into GPRs rD through 31. If the EA is not a multiple of 4 the alignment exception handler may be invoked if a page boundary is crossed.
stmw rS,d{rA)	Store Multiple Word	The effective address is the sum (rAIO)+d. n= (32-rS). n consecutive words starting at the EA are stored from GPRs rS through 31. If the EA is not a multiple of 4 the alignment exception handler may be invoked if a page boundary is crossed.
Opcode	**Integer Move String Instructions**	**Details**
lswi rD,rA,NB	Load String Word Immediate	The EA is (rAIO). Let n = NB if NB:;t:O, n = 32 if NB=O; n is the number of bytes to load. Let nr= (nl4); nris the number of registers to receive data. n consecutive bytes starting at the EA are loaded into GPRs rD through rD+nr-1. Bytes are loaded left to right in each register. The sequence of registers wraps around to rO if required. If the four bytes of register rD+nr-1 are only partially filled, the unfilled low-order byte(s) of that register are cleared to 0. If rA is in the range of registers specified to be loaded, it will be skipped in the load process. If operand rA=O, the register is not considered as used for addressing, and will be loaded.
lswx rD,rA,rB	Load String Word Indexed	The EA is the sum (rAIO)+(rB). Let n = XER[25-31]; n is the number of bytes to load. Let nr = CEIL(nl4); nris the number of registers to receive data. If n>O, n consecutive bytes starting at the EA are loaded into registers rD through rD+nr-1. Bytes are loaded left to right in each register. The sequence of registers wraps around to rO if required. If the four bytes of register rD+nr-1 are only partially filled, the unfilled low-order byte(s) of that register are cleared to 0. If n=O, the contents of register rD is undefined. If rA is in the range of registers specified to be loaded, it will be skipped in the load process. If operand rA=O, the register is not considered as used for addressing, and will be loaded.
lscbx rD,rA,rB	Load String and Compare Byte Indexed	The EA is the sum (rAIO)+(rB). XER[25-31] contains the byte count. Register rD is the starting register. n=XER[25-31], which is the number of bytes to be loaded. nr=CEIL(nl4), which is the number of registers to receive data. Starting with the leftmost byte in rD, consecutive bytes in storage addressed by the EA are loaded into rD through rD+nr-1, wrapping around back through GPR ° if required, until either a byte match is found with XER[16-23] or n bytes have been loaded. If a byte match is found, that byte is also loaded. Bytes are always loaded left to right in the register. In the case when a match was found before n bytes were loaded, the contents of the rightmost byte(s) not loaded of that register and the contents of all succeeding registers up to and including rD+nr-1 are undefined. Also, no reference is made to storage after the matched byte is found. In the case when a match was not found, the contents of the rightmost byte(s) not loaded of rD+nr-1 is undefined. When XER[25-31]=O, the content of rD is unchanged. The count of the number of bytes loaded up to and including the matched byte, if a match was found, is placed in XER[25-31].
lscbx. rD,rA,rB	Load String and Compare Byte Indexed with CR Update.	The dot suffix enables the update of the condition register.
stswi rS,rA,NB	Store String Word Immediate	The EA is (rAIO). Let n = NB if NB*O, n = 32 if NB=O; n is the number of bytes to store. Let nr = CEIL(nl4); nris the number of registers to supply data. n consecutive bytes starting at the EA are stored from register rS through rS+nr-1. Bytes are stored left to right from each register. The sequence of registers wraps around through rO if required.
stswx rS,rA,rB	Store String Word Indexed	The effective address is the sum (rAIO)+(rB). Let n = XER[25-31]; n is the number of bytes to store. Let nr= CEIL(n/4); nris the number of registers to supply data. n consecutive bytes starting at the EA are stored from register rS through rS+nr-1. Bytes are stored left to right from each register. The sequence of registers wraps around through r0 if required.
Opcode	**Memory Synchronization Instructions**	**Details**

eieio	Enforce In·Order Execution of I/O	The eieio instruction provides an ordering function for the effects of load and store instructions executed by a given processor. Executing an eieio instruction ensures that all memory accesses previously initiated by the given processor are complete with respect to main memory before allowing any memory accesses subsequently initiated by the given processor to access main memory. The eieio instruction orders load and store operations to cache inhibited memory, and store operations to write through cache memory. The eieio instruction performs the same function as a sync instruction when executed by the MPC601.
isync	Instruction Synchronize	This instruction waits for all previous instructions to complete, and then discards any prefetched instructions, causing subsequent instructions to be fetched (or refetched) from memory and to execute in the context established by the previous instructions. This instruction has no effect on other processors or on their caches.
lwarx rD,rA,rB	Load Word and Reserve Indexed	The effective address is the sum (rAIO)+(rB). The word in memory addressed by the EA is loaded into register rD. This instruction creates a reservation for use by a stwcx. instruction. An address computed from the EA is associated with the reservation, and replaces any address previously associated with the reservation. The EA must be a multiple of 4. If it is not, the alignment exception handler will be invoked if the word loaded crosses a page boundary, or the results may be undefined.
stwcx. rS,rA,rB	Store Word Conditional Indexed	The effective address is the sum (rAIO)+(rB). If a reservation exists, register rS is stored into the word in memory addressed by the EA and the reservation is cleared. If a reservation does not exist, the instruction completes without altering memory. The EO bit in the condition register field CRO is modified to reflect whether the store operation was performed (Le., whether a reservation existed when the stwcx. instruction began execution). If the store was completed successfully, the EO bit is set to one. The EA must be a multiple of 4; otherwise, the alignment exception handler will be invoked if the word stored crosses a page boundary, or the results may be undefined.
sync	Synchronize	Executing a sync instruction ensures that all instructions previously initiated by the given processor appear to have completed before any subsequent instructions are initiated by the given processor. When the sync instruction completes, all memory accesses initiated by the given processor prior to the sync will have been performed with respect to all other mechanisms that access memory. The sync instruction can be used to ensure that the results of all stores into a data structure, performed in a "critical section" of a program, are seen by other processors before the data structure is seen as unlocked. The Enforce In-Order Execution of liD (eieio) instruction may be more appropriate than sync for cases in which the only requirement is to control the order in which memory references are seen by I/O devices.
Opcode	**Branch Instructions**	**Details**
b imm_addr	Branch.	Branch to the address computed as the sum of the immediate address and the address of the current instruction.
ba imm_addr	Branch Absolute.	Branch to the absolute address specified.
bl imm_addr	Branch then Link.	Branch to the address computed as the sum of the immediate address and the address of the current instruction. The instruction address following this instruction is placed into the link register (LR).
bla imm_addr	Branch Absolute then Link.	Branch to the absolute address specified. The instruction address following this instruction is placed into the link register (LR).
bc BO,BI,target_addr	Branch Conditional.	Branch conditionally to the address computed as the sum of the immediate address and the address of the current instruction. The BI operand specifies the bit in the condition register (CR) to be used as the condition of the branch.
bca BO,BI,target_addr	Branch Conditional Absolute.	Branch conditionally to the absolute address specified.
bcl BO,BI,target_addr	Branch Conditional then Link.	Branch conditionally to the address computed as the sum of the immediate address and the address of the current instruction. The instruction address following this instruction is placed into the link register.
bcla BO,BI,target_addr	Branch Conditional Absolute then Link.	Branch conditionally to the absolute address specified. The instruction address following this instruction is placed into the link register.
bclr BO,BI	Branch Conditional to Link Register	Branch Conditional to Link Register. Branch conditionally to the address in the link register. The BI operand specifies the bit in the condition register to be used belrl as the condition of the branch.
bclrl BO,BI	Branch Conditional to Link Register then Link.	Branch conditionally to the address specified in the link register. The instruction address following this instruction is then placed into the link register.
bcctr BO,BI	Branch Conditional to Count Register.	Branch conditionally to the address specified in the count register. The BI operand specifies the bit in the condition register to be used as the condition of the branch. Note: If the "decrement and test CTR" option is specified (BO[2]=0), the instruction form is invalid. For the MPC601 , the decremented count register is tested for zero and branches based on this test, but instruction fetching is directed to the address specified by the non-decremented version of the count register. Use of this invalid form of this instruction is not recommended.
bcctrl BO,BI	Branch Conditional to Count Register then Link.	Branch conditionally to the address specified in the count register. The instruction address following this instruction is placed into the link register.
Opcode	**Condition Register Logical Instructions**	**Details**
crand crbD,crbA,crbB	Condition Register AND	The bit in the condition register specified by crbA is ANDed with the bit in the condition register specified by crbB. The result is placed into the condition register bit specified by crbD.
cror crbD,crbA,crbB	Condition Register OR	The bit in the condition register specified by crbA is ORed with the bit in the condition register specified by crbB. The result is placed into the condition register bit specified by crbD.
crxor crbD,crbA,crbB	Condition Register XOR	The bit in the condition register specified by crbA is XORed with the bit in the condition register specified by crbB. The result is placed into the condition register bit specified by crbD.
crnand crbD,crbA,crbB	Condition Register NAND	The bit in the condition register specified by crbA is ANDed with the bit in the condition register specified by crbB. The complemented result is placed into the condition register bit specified by crbD.
crnor crbD,crbA,crbB	Condition Register NOR	The bit in the condition register specified by crbA is ORed with the bit in the condition register specified by crbB. The complemented result is placed into the condition register bit specified by crbD.
creqv crbD,crbA,crbB	Condition Register Equivalent	The bit in the condition register specified by crbA is XORed with the bit in the condition register specified by crbB. The complemented result is placed into the condition register bit specified by crbD.
crandc crbD,crbA,crbB	Condition Register AND with Complement	The bit in the condition register specified by crbA is ANDed with the complement of the bit in the condition register specified by crbB and the result is placed into the condition register bit specified by crbD.
crorc crbD,crbA,crbB	Condition Register OR with Complement	The bit in the condition register specified by crbA is ORed with the complement of the bit in the condition register specified by crbB and the result is placed into the condition register bit specified by crbD.
mcrf crfD,crfS	Move Condition Register Field	The contents of crfS are copied into crfD. No other condition register fields are changed.
Opcode	**System Linkage Instructions**	**Details**
sc	System Call	When executed, the effective address of the instruction following the sc instruction is placed into SRR0. Bits 16–31 of the MSR are placed into bits 16–31 of SRR1, and bits 0–15 of SRR1 are set to undefined values. Then a system call exception is generated. The exception causes the MSR to be altered as described in Section 5.4, "Exception Definitions." The exception causes the next instruction to be fetched from offset x'C00' from the base physical address indicated by the new setting of MSR[IP]. For a discussion of POWER compatibility with respect to instruction bits 16–29, refer to Appendix B, Section B.10, "System Call/Supervisor Call." To ensure compatibility with future versions of the PowerPC architecture, bits 16–29 should be coded as zero and bit 30 should be coded as a 1. The PowerPC architecture defines bit 31 as reserved, and thereby cleared to 0; in order for the 601 to maintain compatibility with the POWER architecture, the execution of an sc instruction with bit 31 (the LK bit) set to 1 will cause an update of the Link register with the address of the instruction following the sc instruction. This instruction is context synchronizing.
rfi	Return from Interrupt	Bits 16–31 of SRR1 are placed into bits 16–31 of the MSR, then the next instruction is fetched, under control of the new MSR value, from the address SRR0[0–29] || b'00'. This instruction is a supervisor-level instruction and is context synchronizing.
Opcode	**Trap Instructions and Mnemonics**	**Details**
twi TO,rA,SIMM	Trap Word Immediate	The contents of rA is compared with the sign-extended SIMM operand. If any bit in the TO operand is set to 1 and its corresponding condition is met by the result of the comparison, then the system trap handler is invoked.
tw TO,rA,rB	Trap Word	The contents of rA is compared with the contents of rB. If any bit in the TO operand is set to 1 and its corresponding condition is met by the result of the comparison, then the system trap handler is invoked.
Opcode	**Move to/from Machine State Register/Condition Register Instructions**	**Details**
mtcrf CRM,rS	Move to Condition Register Fields	The contents of rS are placed into the condition register under control of the field mask specified by operand CRM. The field mask identifies the 4-bit fields affected. Let i be an integer in the range 0–7. If CRM(i) = 1, then CR field i (CR bits 4* i through 4* i +3) is set to the contents of the corresponding field of r S. In some PowerPC implementations, this instruction may perform more slowly when only a portion of the fields are updated as opposed to all of the fields. This is not true for the 601.
mcrxr crfD	Move to Condition Register from XER	The contents of XER[0–3] are copied into the condition register field designated by crfD. All other fields of the condition register remain unchanged. XER[0–3] is cleared to 0.
mfcr rD	Move from Condition Register	The contents of the condition register are placed into rD.
mtmsr rS	Move to Machine State Register	The contents of rS are placed into the MSR. This instruction is a supervisor-level instruction and is context synchronizing.
mfmsr rD	Move from Machine State Register	The contents of the MSR are placed into rD. This is a supervisor-level instruction.
Opcode	**Move to/from Special Purpose Register Instructions**	**Details**
mtspr SPR,rS	Move to Special Purpose Register	The SPR field denotes a special purpose register. The contents of rS are placed into the designated SPR. Simplified mnemonic examples: mtxer rA mtspr 1,rA mtlr rA mtspr 8,rA mtctr rA mtspr 9,rA
mfspr rD,SPR	Move from Special Purpose Register	The SPR field denotes a special purpose register. The contents of the designated SPR are placed into rD. Simplified mnemonic examples: mfxer rA mfspr rA,1 mflr rA mfspr rA,8 mfctr rA mfspr rA,9
Opcode	**Cache Management Supervisor-Level Instruction**	**Details**
dcbi rA,rB	Data Cache Block Invalidate	The effective address is the sum (rA|0)+(rB). The action taken depends on the memory mode associated with the target, and the state (modified, unmodified) of the block. The following list describes the action to take if the block containing the byte addressed by the EA is or is not in the cache. • Coherency required (WIM = xx1) — Unmodified block—Invalidates copies of the block in the caches of all processors. — Modified block—Invalidates copies of the block in the caches of all processors. (Discards the modified contents.) — Absent block—If copies are in the caches of any other processor, causes the copies to be invalidated. (Discards any modified contents.) • Coherency not required (WIM = xx0) — Unmodified block—Invalidates the block in the local cache. — Modified block—Invalidates the block in the local cache. (Discards the modified contents.) — Absent block—No action is taken. When data address translation is enabled, MSR[DT]=1, and the logical (effective) address has no translation, a data access exception occurs. See Section 5.4.3, "Data Access Exception (x'00300')." The function of this instruction is independent of the write-through and cache-inhibited/allowed modes determined by the WIM bit settings of the block containing the byte addressed by the EA. This instruction is treated as a store to the addressed byte with respect to address translation and protection. The reference and change bits are modified appropriately. If the EA specifies a memory address for which T = 1 in the corresponding segment register, the instruction is treated as a no-op. This is a supervisor-level instruction.
Opcode	**User-Level Cache Instructions**	**Details**
dcbt rA,rB	Data Cache Block Touch	The EA is the sum (rA|0)+(rB). This instruction provides a method for improving performance through the use of software-initiated fetch hints. The 601 performs the fetch for the cases when the address hits in the UTLB or the BTLB, and when it is permitted load access from the addressed page. The operation is treated similarly to a byte load operation with respect to memory protection. If the address translation does not hit in the UTLB or BTLB, or if it does not have load access permission, the instruction is treated as a no-op. If the access is directed to a cache-inhibited page, or to an I/O controller interface segment, then the bus operation occurs, but the cache is not updated. This instruction never affects the reference or change bits in the hashed page table. While the 601 maintains a cache line size of 64 bytes, the dcbt instruction may only result in the fetch of a 32-byte sector (the one directly addressed by the EA). The other 32-byte sector in the cache line may or may not be fetched, depending on activity in the dynamic memory queue. A successful dcbt instruction will affect the state of the TLB and cache LRU bits as defined by the LRU algorithm.
dcbtst rA,rB	Data Cache Block Touch for Store	The EA is the sum (rA|0)+(rB). The dcbtst instruction operates exactly like the dcbt instruction as implemented on the 601.
clcs rD,rA	Cache Line Compute Size	This is a POWER instruction, and is not part of the PowerPC architecture. This instruction will not be supported by other PowerPC implementations. This instruction places the cache line size specified by operand rA into register rD. The rA operand is encoded as follows: 01100 Instruction cache line size (returns value of 64) 01101 Data cache line size (returns value of 64) 01110 Minimum line size (returns value of 64) 01111 Maximum line size (returns value of 64) All other encodings of the rA operand return undefined values. This instruction is specific to the 601.
dcbz rA,rB	Data Cache Block Set to Zero	The EA is the sum (rA|0)+(rB). If the block (the cache sector consisting of 32 bytes) containing the byte addressed by the EA is in the data cache, all bytes are cleared to 0. If the block containing the byte addressed by the EA is not in the data cache and the corresponding page is caching-allowed, the block is established in the data cache without fetching the block from main memory, and all bytes of the block are cleared to 0. If the page containing the byte addressed by the EA is caching-inhibited or write-through, then the system alignment exception handler is invoked. If the block containing the byte addressed by the EA is in coherence required mode, and the block exists in the data cache(s) of any other processor(s), it is kept coherent in those caches. The dcbz instruction is treated as a store to the addressed byte with respect to address translation and protection. If the EA corresponds to an I/O controller interface segment
dcbst rA,rB	Data Cache Block Store	The EA is the sum(rA|0)+(rB). If the block (the cache sector consisting of 32 bytes) containing the byte addressed by the EA is in coherence required mode, and a block containing the byte addressed by the EA is in the data cache of any processor and has been modified, the writing of it to main memory is initiated. The function of this instruction is independent of the write-through and cache-inhibited/allowed modes of the block containing the byte addressed by the EA. This instruction is treated as a load from the addressed byte with respect to address translation and protection. If the EA corresponds to an I/O controller interface segment (SR[T] = 1), the dcbst instruction is treated as a no-op.
dcbf rA,rB	Data Cache Block Flush	The EA is the sum (rA|0) + (rB). The action taken depends on the memory mode associated with the target, and on the state of the block. The following list describes the action taken for the various cases, regardless of whether the page or block containing the addressed byte is designated as write-through or if it is in the caching-inhibited or caching-allowed mode. • Coherency required (WIM = xx1) — Unmodified block—Invalidates copies of the block in the caches of all processors. — Modified block—Copies the block to memory. Invalidates copies of the block in the caches of all processors. — Absent block—If modified copies of the block are in the caches of other processors, causes them to be copied to memory and invalidated. If unmodified copies are in the caches of other processors, causes those copies to be invalidated. • Coherency not required (WIM = xx0) — Unmodified block—Invalidates the block in the processor's cache. — Modified block—Copies the block to memory. Invalidates the block in the processor's cache. — Absent block—Does nothing.
Opcode	**Segment Register Manipulation Instructions**	**Details**
mtsr SR,rS	Move to Segment Register	The contents of rS is placed into segment register specified by operand SR. This is a supervisor-level instruction.
mtsrin rS,rB	Move to Segment Register Indirect	The contents of rS are copied to the segment register selected by bits 0–3 of rB. This is a supervisor-level instruction.
mfsr rD,SR	Move from Segment Register	The contents of the segment register specified by operand SR are placed into rD. This is a supervisor-level instruction.
mfsrin rD,rB	Move from Segment Register Indirect	The contents of the segment register selected by bits 0–3 of rB are copied into rD. This is a supervisor-level instruction.
Opcode	**Translation Lookaside Buffer Management Instruction**	**Details**

tlbie rB	Translation Lookaside Buffer Invalidate Entry	The effective address is the contents of rB. If the TLB contains an entry corresponding to the EA, that entry is removed from the TLB. The TLB search is done regardless of the settings of MSR[IT] and MSR[DT]. Also, a TLB invalidate operation is broadcast on the system bus unless disabled by setting bit 17 in HID1. Block address translation for the EA, if any, is ignored. Because the 601 supports broadcast of TLB entry invalidate operations, the following must be observed: • The tlbie instruction must be contained in a critical section of memory controlled by software locking, so that the tlbie is issued on only one processor at a time. • A sync instruction must be issued after every tlbie and at the end of the critical section. This causes hardware to wait for the effects of the preceding tlbie instructions(s) to propagate to all processors. A processor detecting a TLB invalidate broadcast does the following: 1. Prevents execution of any new load, store, cache control or tlbie instructions and prevents any new reference or change bit updates 2. Waits for completion of any outstanding memory operations (including updates to the reference and change bits associated with the entry to be invalidated) 3. Invalidates the two entries (both associativity classes) in the UTLB indexed by the matching address 4. Resumes normal execution This is a supervisor-level instruction. Nothing is guaranteed about instruction fetching in other processors if tlbie deletes the page in which another processor is executing.

Opcode	External Control Instructions	Details
eciwx rD,rA,rB	External Control Input Word Indexed	The EA is the sum (rA|0) + (rB). If the external access register (EAR) E-bit (bit 0) is set to 1, a load request for the physical address corresponding to the EA is sent to the device identified by the EAR Resource ID bits (bits 28–31), bypassing the cache. The word returned by the device is placed in rD. The EA sent to the device must be word aligned. If the EAR[E] = 0, a data access exception is invoked, with bit 11 of DSISR set to 1, and bit 6 cleared to 0 to indicate that the exception occurred during a load operation. The eciwx instruction is supported for EAs that reference ordinary memory segments (SR[T] = 0), for EAs mapped by BAT registers, and for EAs generated when MSR[DT] = 0.The instruction is treated as a no-op for EAs in I/O controller interface segments (SR[T] = 1). The access caused by this instruction is treated as a load from the location addressed by the EA with respect to protection and reference and change recording.
ecowx rS,rA,rB	External Control Output Word Indexed	The EA is the sum (rA|0) + (rB). If the External Access Register (EAR) E-bit (bit 0) is set to 1, a store request for the physical address corresponding to the EA and the contents of rS are sent to the device identified by EAR[RID] (resource ID) (bits 28–31), bypassing the cache. The EA sent to the device must be word aligned. If the EAR[E] = 0, a data access exception is invoked, with bit 11 of DSISR set to 1, and bit 6 set to 1 to indicate that the exception occurred during a store operation. The ecowx instruction is supported for EAs that reference ordinary memory segments (SR[T] = 0), for EAs mapped by BAT registers, and for EAs generated when MSR[DT] = 0.The instruction is treated as a no-op for EAs in I/O controller interface segments (SR[T] = 1). The access caused by this instruction is treated as a store to the location addressed by the EA with respect to protection and reference and change recording

Opcode	Miscellaneous Simplified Mnemonics	Details
no-op	No-Op	(equivalent to ori 0,0,0)
li rD,value	Load Immediate	Load a 16-bit signed immediate value into rA (equivalent to addi rA,0,value)
lis rD,value	Load Shifted Immediate	Load a 16-bit signed immediate value, shifted left by 16 bits, into rA (equivalent to addis rA,0,value)
la rD,SIMM(rA)	Load Address	equivalent to addi rD,rA,SIMM
la rD,v	Load Address	If the variable v is located at offset SIMMv bytes from the address in register rv, and the assembler has been told to use register rv as a base for references to the data structure containing v, then the following line causes the address of v to be loaded into register rD. (equivalent to addi rD,rA,SIMMv)
mr rA,rS	Move Register	(equivalent to or rA,rS,rS)
not rA,rS	Complement Register	(equivalent to nor rA,rS,rS)

Opcode	Floating-Point Arithmetic Instructions	Details
fadd frD,frA,frB	Floating-Point Add	The floating-point operand in register frA is added to the floating-point operand in register frB. If the most significant bit of the resultant significand is not a one the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into register frD. Floating-point addition is based on exponent comparison and addition of the two significands. The exponents of the two operands are compared, and the significand accompanying the smaller exponent is shifted right, with its exponent increased by one for each bit shifted, until the two exponents are equal. The two significands are then added algebraically to form an intermediate sum. All 53 bits in the significand as well as all three guard bits (G, R, and X) enter into the computation. If a carry occurs, the sum's significand is shifted right one bit position and the exponent is increased by one. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.
fadd. frD,frA,frB	Floating-Point Add with CR Update.	The dot suffix enables the update of the condition register.
fadds frD,frA,frB	Floating-Point Add Single-Precision	The floating-point operand in register frA is added to the floating-point operand in register frB. If the most significant bit of the resultant significand is not a one, the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into register frD. Floating-point addition is based on exponent comparison and addition of the two significands. The exponents of the two operands are compared, and the significand accompanying the smaller exponent is shifted right, with its exponent increased by one for each bit shifted, until the two exponents are equal. The two significands are then added algebraically to form an intermediate sum. All 53 bits in the significand as well as all three guard bits (G, R, and X) enter into the computation. If a carry occurs, the sum's significand is shifted right one bit position and the exponent is increased by one. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.
fadds. frD,frA,frB	Floating-Point Single-Precision with CR Update.	The dot suffix enables the update of the condition register.
fsub frD,frA,frB	Floating-Point Subtract	The floating-point operand in register frB is subtracted from the floating-point operand in register frA. If the most significant bit of the resultant significand is not a 1, the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into register frD. The execution of the Floating-Point Subtract instruction is identical to that of Floating-Point Add, except that the contents of register frB participates in the operation with its sign bit (bit 0) inverted. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.
fsub. frD,frA,frB	Floating-Point Subtract with CR Update.	The dot suffix enables the update of the condition register.
fsubs frD,frA,frB	Floating-Point Subtract Single-Precision	The floating-point operand in register frB is subtracted from the floating-point operand in register frA. If the most significant bit of the resultant significand is not a 1, the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into register frD. The execution of the Floating-Point Subtract instruction is identical to that of Floating-Point Add, except that the contents of register frB participates in the operation with its sign bit (bit 0) inverted. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.
fsubs. frD,frA,frB	Floating-Point Subtract Single-Precision with CR Update.	The dot suffix enables the update of the condition register.
fmul frD,frA,frC	Floating-Point Multiply	The floating-point operand in register frA is multiplied by the floating-point operand in register frC. If the most significant bit of the resultant significand is not a 1, the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into register frD. Floating-point multiplication is based on exponent addition and multiplication of the significands. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.
fmul. frD,frA,frC	Floating-Point Multiply with CR Update.	The dot suffix enables the update of the condition register.
fmuls frD,frA,frC	Floating-Point Multiply Single-Precision	The floating-point operand in register frA is multiplied by the floating-point operand in register frC. If the most significant bit of the resultant significand is not a 1, the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into register frD. Floating-point multiplication is based on exponent addition and multiplication of the significands. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.
fmuls. frD,frA,frC	Floating-Point Multiply Single-Precision with CR Update.	The dot suffix enables the update of the condition register.
fdiv frD,frA,frB	Floating-Point Divide	The floating-point operand in register frA is divided by the floating-point operand in register frB. No remainder is preserved. If the most significant bit of the resultant significand is not a 1, the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into register frD. Floating-point division is based on exponent subtraction and division of the significands. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1 and zero divide exceptions when FPSCR[ZE] = 1.
fdiv. frD,frA,frB	Floating-Point Divide with CR Update.	The dot suffix enables the update of the condition register.
fdivs frD,frA,frB	Floating-Point Divide Single-Precision	The floating-point operand in register frA is divided by the floating-point operand in register frB. No remainder is preserved. If the most significant bit of the resultant significand is not a 1, the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into register frD. Floating-point division is based on exponent subtraction and division of the significands. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1 and zero divide exceptions when FPSCR[ZE] = 1.
fdivs. frD,frA,frB	Floating-Point Divide Single-Precision with CR Update.	The dot suffix enables the update of the condition register.

Opcode	Floating-Point Multiply-Add Instructions	Details
fmadd frD,frA,frC,frB	Floating-Point Multiply-Add	The floating-point operand in register frA is multiplied by the floating-point operand in register frC. The floating-point operand in register frB is added to this intermediate result. If the most significant bit of the resultant significand is not a one the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into register frD. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1
fmadd. frD,frA,frC,frB	Floating-Point Multiply-Add with CR Update.	The dot suffix enables the update of the condition register.
fmadds frD,frA,frC,frB	Floating-Point Multiply-Add Single- Precision	The floating-point operand in register frA is multiplied by the floating-point operand in register frC. The floating-point operand in register frB is added to this intermediate result. If the most significant bit of the resultant significand is not a one the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into register frD. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.
fmadds. frD,frA,frC,frB	Floating-Point Multiply-Add Single-Precision with CR Update.	The dot suffix enables the update of the condition register.
fmsub frD,frA,frC,frB	Floating-Point Multiply-Subtract	The floating-point operand in register frA is multiplied by the floating-point operand in register frC. The floating-point operand in register frB is subtracted from this intermediate result. If the most significant bit of the resultant significand is not a one the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into register frD. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.
fmsub. frD,frA,frC,frB	Floating-Point Multiply-Subtract with CR Update.	The dot suffix enables the update of the condition register.
fmsubs frD,frA,frC,frB	Floating-Point Multiply-Subtract Single-Precision	The floating-point operand in register frA is multiplied by the floating-point operand in register frC. The floating-point operand in register frB is subtracted from this intermediate result. If the most significant bit of the resultant significand is not a one the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR and placed into register frD. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1
fmsubs. frD,frA,frC,frB	Floating-Point Multiply-Subtract Single-Precision + CR Udt.	The dot suffix enables the update of the condition register.
fnmadd frD,frA,frC,frB	Floating-Point Negative Multiply-Add	The floating-point operand in register frA is multiplied by the floating-point operand in register frC. The floating-point operand in register frB is added to this intermediate result. If the most significant bit of the resultant significand is not a one the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR, then negated and placed into register frD. This instruction produces the same result as would be obtained by using the floating-point multiply-add instruction and then negating the result, with the following exceptions: • QNaNs propagate with no effect on their sign bit. • QNaNs that are generated as the result of a disabled invalid operation exception have a "sign" bit of zero. • SNaNs that are converted to QNaNs as the result of a disabled invalid operation exception retain the "sign" bit of the SNaN. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1 .
fnmadd. frD,frA,frC,frB	Floating-Point Negative Multiply-Add with CR Update.	The dot suffix enables the update of the condition register.
fnmadds frD,frA,frC,frB	Floating-Point Negative Multiply-Add Single-Precision	The floating-point operand in register frA is multiplied by the floating-point operand in register frC. The floating-point operand in register frB is added to this intermediate result. If the most significant bit of the resultant significand is not a one the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR, then negated and placed into register frD. This instruction produces the same result as would be obtained by using the floating-point multiply-add instruction and then negating the result, with the following exceptions: • QNaNs propagate with no effect on their sign bit. • QNaNs that are generated as the result of a disabled invalid operation exception have a "sign" bit of zero. • SNaNs that are converted to QNaNs as the result of a disabled invalid operation exception retain the "sign" bit of the SNaN. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1 .
fnmadds. frD,frA,frC,frB	Floating-Point Neg Multiply-Add Single-Precision + CR Upd	The dot suffix enables the update of the condition register.
fnmsub frD,frA,frC,frB	Floating-Point Negative Multiply-Subtract	The floating-point operand in register frA is multiplied by the floating-point operand in register frC. The floating-point operand in register frB is subtracted from this intermediate result. If the most significant bit of the resultant significand is not a one the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR, then negated and placed into register frD. This instruction produces the same result as would be obtained by using the floating-point multiply-subtract instruction and then negating the result, with the following exceptions: • QNaNs propagate with no effect on their sign bit. • QNaNs that are generated as the result of a disabled invalid operation exception have a sign bit of zero. • SNaNs that are converted to QNaNs as the result of a disabled invalid operation exception retain the sign bit of the SNaN. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.
fnmsub. frD,frA,frC,frB	Floating-Point Negative Multiply-Subtract with CR Update.	The dot suffix enables the update of the condition register.
fnmsubs frD,frA,frC,frB	Floating-Point Negative Multiply-Subtract Single-Precision	The floating-point operand in register frA is multiplied by the floating-point operand in register frC. The floating-point operand in register frB is subtracted from this intermediate result. If the most significant bit of the resultant significand is not a one the result is normalized. The result is rounded to the target precision under control of the floating-point rounding control field RN of the FPSCR, then negated and placed into register frD. This instruction produces the same result as would be obtained by using the floating-point multiply-subtract instruction and then negating the result, with the following exceptions: • QNaNs propagate with no effect on their "sign" bit. • QNaNs that are generated as the result of a disabled invalid operation exception have a "sign" bit of zero. • SNaNs that are converted to QNaNs as the result of a disabled invalid operation exception retain the "sign" bit of the SNaN. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.
fnmsubs. frD,frA,frC,frB	Single-Precision with CR Update.	The dot suffix enables the update of the condition register.

Opcode	Floating-Point Rounding and Conversion Instructions	Details
frsp frD,frB	Floating-Point Round to Single-Precision	If it is already in single-precision range, the floating-point operand in register frB is placed into register frD. Otherwise the floating-point operand in register frB is rounded to single-precision using the rounding mode specified by FPSCR[RN] and placed into register frD. FPSCR[FPRF] is set to the class and sign of the result, except for invalid operation exceptions when FPSCR[VE] = 1.
frsp. frD,frB	Floating-Point Round to Single-Precision with CR Update.	The dot suffix enables the update of the condition register.
fctiw frD,frB	Floating-Point Convert to Integer Word	The floating-point operand in register frB is converted to a 32-bit signed integer, using the rounding mode specified by FPSCR[RN], and placed in bits 32–63 of register frD. Bits 0–31 of register frD are undefined. If the operand in register frB is greater than 2 31 – 1, bits 32–63 of register frD are set to x'7FFF_FFFF'. If the operand in register frB is less than –2 31 , bits 32–63 of register frD are set to x '8000_0000'.Except for trap-enabled invalid operation exceptions, FPSCR[FPRF] is undefined. FPSCR[FR] is set if the result is incremented when rounded. FPSCR[FI] is set if the result is inexact.
fctiw. frD,frB	Floating-Point Convert to Integer Word with CR Update.	The dot suffix enables the update of the condition register.
fctiwz frD,frB	Floating-Point Convert to Integer Word with Round Toward Zero	The floating-point operand in register frB is converted to a 32-bit signed integer, using the rounding mode Round toward Zero, and placed in bits 32–63 of register frD. Bits 0–31 of register frD are undefined. If the operand in frB is greater than 2 31 – 1, bits 32–63 of frD are set to x'7FFF_FFFF'. If the operand in register frB is less than –2 31 , bits 32–63 of register frD are set to x '8000_0000'. The conversion is described fully in Appendix F, “Floating-Point Models.” Except for trap-enabled invalid operation exceptions, FPSCR[FPRF] is undefined. FPSCR[FR] is set if the result is incremented when rounded. FPSCR[FI] is set if the result is inexact.
fctiwz. frD,frB	Float-Point Conv to Int Word with Rnd To Zero + CR Upd	The dot suffix enables the update of the condition register.

Opcode	Floating-Point Compare Instructions	Details
fcmpu crfD,frA,frB	Floating-Point Compare Unordered	The floating-point operand in register frA is compared to the floating-point operand in register frB. The result of the compare is placed into CR field crfD and the FPCC. If an operand is a NaN, either quiet or signaling, CR field crfD and the FPCC are set to reflect unordered. If an operand is a Signaling NaN,VXSNAN is set.
fcmpo crfD,frA,frB	Floating-Point Compare Ordered	The floating-point operand in register frA is compared to the floating-point operand in register frB. The result of the compare is placed into CR field crfD and the FPCC. If an operand is a NaN, either quiet or signalling, CR field crfD and the FPCC are set to reflect unordered. If an operand is a Signalling NaN, VXSNAN is set, and if invalid operation is disabled (VE = 0) then VXVC is set. Otherwise, if an operand is a Quiet NaN, VXVC is set.

Opcode	Floating-Point Compare Instructions	Details
mffs frD	Move from FPSCR	The contents of the FPSCR are placed into bits 32–63 of register frD.In the 601, bits 0–31 of floating-point register frD are set to the value x'FFFF_FFFF'.
mffs. frD	Move from FPSCR with CR Update.	The dot suffix enables the update of the condition register.
mcrfs crfD,crfS	Move to Condition Register from FPSCR	The contents of FPSCR field specified by operand crfS are copied to the CR field specified by operand crfD. All exception bits copied are cleared to zero in the FPSCR.
mtfsfi crfD,IMM	Move to FPSCR Field Immediate	The value of the IMM field is placed into FPSCR field crfD. All other FPSCR fields are unchanged.When FPSCR[0–3] is specified, bits 0 (FX) and 3 (OX) are set to the values of IMM[0] and IMM[3] (that is, even if this instruction causes OX to change from 0 to 1, FX is set from IMM[0] and not by the usual rule that FX is set to 1 when an exception bit changes from 0 to 1). Bits 1 and 2 (FEX and VX) are set according to the usual rule, and not from IMM[1–2].
mtfsfi. crfD,IMM	Move to FPSCR Field Immediate with CR Update.	The dot suffix enables the update of the condition register.
mtfsf FM,frB	Move to FPSCR Fields	Bits 32–63 of register frB are placed into the FPSCR under control of the field mask specified by FM. The field mask identifies the 4-bit fields affected. Let i be an integer in the range 0–7. If FM = 1 then FPSCR field i (FPSCR bits 4 ∗ i through 4 ∗ i +3) is set to the contents of the corresponding field of the low-order 32 bits of register frB. When FPSCR[0–3] is specified, bits 0 (FX) and 3 (OX) are set to the values of frB[32] and frB[35] (that is, even if this instruction causes OX to change from 0 to 1, FX is set from frB[32] and not by the usual rule that FX is set to 1 when an exception bit changes from 0 to 1). Bits 1 and 2 (FEX and VX) are set according to the usual rule described in Section 2.2.3, “Floating-Point Status and Control Register (FPSCR),” and not from frB[33–34].
mtfsf. FM,frB	Move to FPSCR Fields with CR Update.	The dot suffix enables the update of the condition register. In other PowerPC implementations, the mtfsf instruction may perform more slowly when only a portion of the fields are updated.This is not the case in the 601.
mtfsb0 crbD	Move to FPSCR Bit 0	The bit of the FPSCR specified by operand crbD is cleared to 0. Bits 1 and 2 (FEX and VX) cannot be explicitly reset.
mtfsb0. crbD	Move to FPSCR Bit 0 with CR Update.	The dot suffix enables the update of the condition register.
mtfsb1 crbD	Move to FPSCR Bit 1	The bit of the FPSCR specified by operand crbD is set to 1. Bits 1 and 2 (FEX and VX) cannot be reset explicitly.
mtfsb1. crbD	Move to FPSCR Bit 1 with CR Update.	The dot suffix enables the update of the condition register.

Super-H

Opcode	Instruction	Description	Function	Code	Tbit	Cycl	Example
ADD Rm,Rn	ADD Binary	Adds general register Rn data to Rm data, and stores the result in Rn	Rm + Rn → Rn	0011nnnnmmmm1100	-	1	ADD R0,R1
ADD #imm,Rn	ADD Binary	8-bit immediate data can be added instead of Rm data. Since the 8-bit immediate data is sign-extended to 32 bits, this instruction can add and subtract immediate data.	Rn + #imm → Rn	0111nnnniiiiiiii	-	1	ADD #H'01,R2
ADDC Rm,Rn	ADD with Carry	Adds Rm data and the T bit to general register Rn data, and stores the result in Rn. The T bit changes according to the result.	Rn + Rm + T → Rn, carry → T	0011nnnnmmmm1110	Carry	1	ADDC R3,R1
ADDV Rm,Rn	ADD with V Flag Overflow Check	Adds general register Rn data to Rm data, and stores the result in Rn. If an overflow occurs, the T bit is set to 1.	Rn + Rm → Rn, overflow → T	0011nnnnmmmm1111	Ovfw	1	ADDV R0,R1
AND Rm,Rn	AND Logical	Logically ANDs general registers Rn and Rm, and stores the result in Rn.	Rn & Rm → Rn	0010nnnnmmmm1001	-	1	AND R0,R1
AND #imm,R0	AND Logical	The contents of general register R0 can be ANDed with zero-extended 8-bit immediate	R0 & imm → R0	11001001iiiiiiii	-	1	AND #H'0F,R0
AND.B #imm,@(R0,GBR)	AND Logical	8-bit memory data pointed to by GBR relative addressing can be ANDed with 8-bit immediate data.	(R0 + GBR) & imm → (R0 + GBR)	11001101iiiiiiii	-	3	AND.B #H'80,@(R0,GBR)
BF label	Branch if False	Reads the T bit, and conditionally branches. If T = 0, it branches to the branch destination address. If T = 1, BF executes the next instruction. The branch destination is an address specified by PC + displacement.	When T = 0, disp × 2 + PC → PC; When T = 1, nop	10001011dddddddd	-	3/1	BF TRGET_F
BF/S label	Branch if False with Delay Slot	Reads the T bit and conditionally branches. If T = 0, it branches after executing the next instruction. If T = 1, BF/S executes the next instruction. The branch destination is an address specified by PC + displacement.	When T = 0, disp × 2+ PC → PC; When T = 1, nop	10001111dddddddd	-	2/1	BF/S TRGET_F
BRA label	Branch	Branches unconditionally after executing the instruction following this BRA instruction. The branch destination is an address specified by PC + displacement However, in this case it is used for address calculation.	disp × 2 + PC → PC	1010dddddddddddd	-	2	BRA TRGET
BRAF Rm	Branch Far	Branches unconditionally. The branch destination is PC + the 32-bit contents of the general register Rm.	Rm + PC → PC	0000mmmm00100011	-	2	
BSR label	Branch to Subroutine	Branches to the subroutine procedure at a specified address. The PC value is stored in the PR, and the program branches to an address specified by PC + displacement However, in this case it is used for address calculation.	PC → PR, disp × 2+ PC → PC	1011dddddddddddd	-	2	BSR TRGET
BSRF Rm	Branch to Subroutine Far	Branches to the subroutine procedure at a specified address after executing the instruction following this BSRF instruction. The PC value is stored in the PR.	PC → PR, Rm + PC → PC	0000mmmm00000011	-	2	BRSF R0
BT label	Branch if True	Reads the T bit, and conditionally branches. If T = 1, BT branches. If T = 0, BT executes the next instruction. The branch destination is an address specified by PC + displacement.	When T = 1, disp × 2 + PC → PC; When T = 0, nop	10001001dddddddd	-	3/1	BT TRGET_T
BT/S label	Branch if True with Delay Slot	Reads the T bit and conditionally branches. If T = 1, BT/S branches after the following instruction executes. If T = 0, BT/S executes the next instruction. The branch destination is an address specified by PC + displacement	When T = 1,disp × 2 + PC → PC; When T = 0, nop	10001101dddddddd	-	2/1	BT/S TARGET_T
CLRMAC	Clear MAC Register	Clear the MACH and MACL Register.	0 → MACH, MACL	0000000000101000	-	1	CLRMAC
CLRT	Clear T Bit	Clears the T bit.	0 → T	0000000000001000	-	1	CLRT
CMP/EQ Rm,Rn	Compare Conditionally	If Rn = Rm, T = 1	When Rn = Rm,1 → T	0011nnnnmmmm0000	reslt	1	
CMP/GE Rm,Rn	Compare Conditionally	If Rn • Rm with signed data, T = 1	When signed and Rn • Rm, 1 → T	0011nnnnmmmm0011	reslt	1	CMP/GE R0,R1
CMP/GT Rm,Rn	Compare Conditionally	If Rn > Rm with signed data, T = 1	When signed and Rn > Rm, 1 → T	0011nnnnmmmm0111	reslt	1	
CMP/HI Rm,Rn	Compare Conditionally	If Rn > Rm with unsigned data, T = 1	When unsigned and Rn > Rm, 1 → T	0011nnnnmmmm0110	reslt	1	
CMP/HS Rm,Rn	Compare Conditionally	If Rn • Rm with unsigned data, T = 1	When unsigned and Rn • Rm, 1 → T	0011nnnnmmmm0010	reslt	1	CMP/HS R0,R1
CMP/PL Rn	Compare Conditionally	If Rn > 0, T = 1	When Rn > 0, 1 → T	0100nnnn00010101	reslt	1	
CMP/PZ Rn	Compare Conditionally	If Rn • 0, T = 1	When Rn • 0, 1 → T	0100nnnn00010001	reslt	1	
CMP/STR Rm,Rn	Compare Conditionally	If a byte in Rn equals a byte in Rm, T = 1	When byte in Rn = byte in Rm, 1 → T	0010nnnnmmmm1100	reslt	1	CMP/STR R2,R3
CMP/EQ #imm,R0	Compare Conditionally	If R0 = imm, T = 1	When R0 = imm, 1 → T	10001000iiiiiiii	reslt	1	
DIV0S Rm,Rn	Divide Step 0 as Signed	DIV0S is an initialization instruction for signed division. It finds the quotient by repeatedly dividing in combination with the DIV1 or another instruction that divides for each bit after this instruction.	MSB of Rn → Q, MSB of Rm → M,M^Q → T	0010nnnnmmmm0111	reslt	1	DIV0S R0,R1
DIV0U	Divide Step 0 as Unsigned	DIV0U is an initialization instruction for unsigned division. It finds the quotient by repeatedly dividing in combination with the DIV1 or another instruction that divides for each bit after this instruction.	0 → M/Q/T	0000000000011001	0	1	DIV0U
DIV1 Rm,Rn	Divide 1 Step	Uses single-step division to divide one bit of the 32-bit data in general register Rn (dividend) by Rm data (divisor). It finds a quotient through repetition either independently or used in combination with other instructions. During this repetition, do not rewrite the specified register or the M, Q, and T bits.	1 step division (Rn ÷ Rm)	0011nnnnmmmm0100	reslt	1	DIV1 R0,R1
DMULS.L Rm,Rn	Double-Length Multiply as Signed	Performs 32-bit multiplication of the contents of general registers Rn and Rm, and stores the 64-bit results in the MACL and MACH register. The operation is a signed arithmetic operation.	With sign,Rn × Rm →MACH, MACL	0011nnnnmmmm1101	-	2-4	DMULS.L R0,R1
DMULU.L Rm,Rn	Double-Length Multiply as Unsigned	Performs 32-bit multiplication of the contents of general registers Rn and Rm, and stores the 64-bit results in the MACL and MACH register. The operation is an unsigned arithmetic operation.	Without sign,Rn × Rm →MACH, MACL	0011nnnnmmmm0101	-	2-4	DMULU.L R0,R1
DT Rn	Decrement and Test	The contents of general register Rn are decremented by 1 and the result compared to 0 (zero). When the result is 0, the T bit is set to 1. When the result is not zero, the T bit is set to 0.	Rn – 1 → Rn;When Rn is 0,1 → T, when Rn is nonzero, 0 → T	0100nnnn00010000	reslt	1	DT R5
EXTS.B Rm,Rn	Extend as Signed	Sign-extends general register Rm data, and stores the result in Rn	Sign-extend Rm from byte → Rn	0110nnnnmmmm1110	-	1	EXTS.B R0,R1
EXTS.W Rm,Rn	Extend as Signed	Sign-extends general register Rm data, and stores the result in Rn	Sign-extend Rm from word → Rn	0110nnnnmmmm1111	-	1	EXTS.W R0,R1
EXTU.B Rm,Rn	Extend as Unsigned	Zero-extends general register Rm data, and stores the result in Rn.	Zero-extend Rm from byte → Rn	0110nnnnmmmm1100	-	1	EXTU.B R0,R1
EXTU.W Rm,Rn	Extend as Unsigned	Zero-extends general register Rm data, and stores the result in Rn.	Zero-extend Rm from word → Rn	0110nnnnmmmm1101	-	1	EXTU.W R0,R1
JMP @Rm	Jump	Branches unconditionally to the address specified by register indirect addressing. The branch destination is an address specified by the 32-bit data in general register Rm.	Rm → PC	0100mmmm00101011	-	2	JMP @R0
JSR @Rm	Jump to Subroutine	Branches to the subroutine procedure at the address specified by register indirect addressing. The PC value is stored in the PR. The jump destination is an address specified by the 32-bit data in general register Rm. The stored/saved PC is the address four bytes after this instruction.	PC → PR, Rm → PC	0100mmmm00001011	-	2	JSR @R0
LDC Rm,SR	Load to Control Register	Store the source operand into control register	Rm → SR	0100mmmm00001110	LSB	1	LDC R0,SR
LDC Rm,GBR	Load to Control Register	Store the source operand into control register	Rm → GBR	0100mmmm00011110	-	1	
LDC Rm,VBR	Load to Control Register	Store the source operand into control register	Rm → VBR	0100mmmm00101110	-	1	
LDC Rm,MOD	Load to Control Register	Store the source operand into control register	Rm → MOD	0100mmmm01011110	-	1	
LDC Rm,RE	Load to Control Register	Store the source operand into control register	Rm → RE	0100mmmm01111110	-	1	
LDC Rm,RS	Load to Control Register	Store the source operand into control register	Rm → RS	0100mmmm01101110	-	1	
LDC.L @Rm+,SR	Load to Control Register	Store the source operand into control register	(Rm) → SR, Rm + 4 → Rm	0100mmmm00000111	LSB	3	
LDC.L @Rm+,GBR	Load to Control Register	Store the source operand into control register	(Rm) → GBR, Rm + 4 → Rm	0100mmmm00010111	-	3	LDC.L @R15+,GBR
LDC.L @Rm+,VBR	Load to Control Register	Store the source operand into control register	(Rm) → VBR, Rm + 4 → Rm	0100mmmm00100111	-	3	
LDC.L @Rm+,MOD	Load to Control Register	Store the source operand into control register	(Rm) → MOD, Rm + 4 → Rm	0100mmmm01010111	-	3	
LDC.L @Rm+,RE	Load to Control Register	Store the source operand into control register	(Rm) → RE, Rm + 4 → Rm	0100mmmm01110111	-	3	
LDC.L @Rm+,RS	Load to Control Register	Store the source operand into control register	(Rm) → RS, Rm + 4 → Rm	0100mmmm01100111	-	3	
LDRE @(disp,PC)	Load Effective Address to RE Register	Stores the effective address of the source operand in the repeat end register RE. The effective address is an address specified by PC + displacement.	disp × 2 + PC → RE	10001110dddddddd	-	1	LDRE END
LDRS @(disp,PC)	Load Effective Address to RS Register	Stores the effective address of the source operand in the repeat start register RS. The effective address is an address specified by PC + displacement.	disp × 2 + PC→ RS	10001100dddddddd	-	1	LDRS STA
LDS Rm,MACH	Load to System Register	Store the source operand into the system register MACH, MACL, or PR or the DSP register DSR, A0, X0, X1, Y0, or Y1. When A0 is designated as the destination, the MSB of the data is copied into A0G.	Rm → MACH	0100mmmm00001010	-	1	
LDS Rm,MACL	Load to System Register		Rm → MACL	0100mmmm00011010	-	1	
LDS Rm,PR	Load to System Register		Rm → PR	0100mmmm00101010	-	1	LDS R0,PR
LDS Rm,DSR	Load to System Register		Rm → DSR	0100mmmm01101010	-	1	
LDS Rm,A0	Load to System Register		Rm → A0	0100mmmm01111010	-	1	
LDS Rm,X0	Load to System Register		Rm → X0	0100mmmm10001010	-	1	
LDS Rm,X1	Load to System Register		Rm → X1	0100mmmm10011010	-	1	
LDS Rm,Y0	Load to System Register		Rm → Y0	0100mmmm10101010	-	1	
LDS Rm,Y1	Load to System Register		Rm → Y1	0100mmmm10111010	-	1	
LDS.L @Rm+,MACH	Load to System Register	Store the source operand into the system register MACH, MACL, or PR or the DSP register DSR, A0, X0, X1, Y0, or Y1. When A0 is designated as the destination, the MSB of the data is copied into A0G.	(Rm) → MACH,Rm + 4 → Rm	0100mmmm00000110	-	1	
LDS.L @Rm+,MACL	Load to System Register		(Rm) → MACL,Rm + 4 → Rm	0100mmmm00010110	-	1	LDS.L @R15+,MACL
LDS.L @Rm+,PR	Load to System Register		(Rm) → PR,Rm + 4 → Rm	0100mmmm00100110	-	1	
LDS.L @Rm+,DSR	Load to System Register		(Rm) → DSR,Rm + 4 → Rm	0100mmmm01100110	-	1	
LDS.L @Rm+,A0	Load to System Register		(Rm) → A0,Rm + 4 → Rm	0100mmmm01110110	-	1	
LDS.L @Rm+,X0	Load to System Register		(Rm) → X0,Rm+4 → Rm	0100nnnn10000110	-	1	
LDS.L @Rm+,X1	Load to System Register		(Rm) → X1,Rm+4 → Rm	0100nnnn10010110	-	1	
LDS.L @Rm+,Y0	Load to System Register		(Rm) → Y0,Rm+4 → Rm	0100nnnn10100110	-	1	
LDS.L @Rm+,Y1	Load to System Register		(Rm) → Y1, Rm+4 → Rm	0100nnnn10110110	-	1	
MAC.L @Rm+,@Rn+	Multiply and Accumulate Calculation Long	Does signed multiplication of 32-bit operands obtained using the contents of general registers Rm and Rn as addresses. The 64-bit result is added to contents of the MAC register, and the final result is stored in the MAC register. Every time an operand is read, they increment Rm and Rn by four.	Signed operation (Rn) × (Rm) + MAC→ MAC	0000nnnnmmmm1111	-	3/2-4	MAC.L @R0+,@R1+
MAC.W @Rm+,@Rn+ **MAC** @Rm+,@Rn+	Multiply and Accumulate Calculation Word	Does signed multiplication of 16-bit operands obtained using the contents of general registers Rm and Rn as addresses. The 32-bit result is added to contents of the MAC register, and the final result is stored in the MAC register. Rm and Rn data are incremented by 2 after the operation.	With sign, (Rn) × (Rm) + MAC → MAC	0100nnnnmmmm1111	-	3/2-4	MAC.W @R0+,@R1+
MOV Rm,Rn	Move Data	Transfers the source operand to the destination. When the operand is stored in memory, the transferred data can be a byte, word, or longword. Loaded data from memory is stored in a register after it is sign-extended to a longword.	Rm → Rn	0110nnnnmmmm0011	-	1	MOV R0,R1
MOV.B Rm,@Rn	Move Data		Rm → (Rn)	0010nnnnmmmm0000	-	1	
MOV.W Rm,@Rn	Move Data		Rm → (Rn)	0010nnnnmmmm0001	-	1	MOV.W R0,@R1
MOV.L Rm,@Rn	Move Data		Rm → (Rn)	0010nnnnmmmm0010	-	1	
MOV.B @Rm,Rn	Move Data		(Rm) → sign extension → Rn	0110nnnnmmmm0000	-	1	
MOV.W @Rm,Rn	Move Data		(Rm) → sign extension → Rn	0110nnnnmmmm0001	-	1	
MOV.L @Rm,Rn	Move Data		(Rm) → Rn	0110nnnnmmmm0010	-	1	
MOV.B Rm,@–Rn	Move Data		Rn – 1 → Rn,Rm → (Rn)	0010nnnnmmmm0100	-	1	
MOV.W Rm,@–Rn	Move Data		Rn – 2 → Rn,Rm → (Rn)	0010nnnnmmmm0101	-	1	MOV.W R0,@–R1
MOV.L Rm,@–Rn	Move Data		Rn – 4 → Rn,Rm → (Rn)	0010nnnnmmmm0110	-	1	
MOV.B @Rm+,Rn	Move Data		(Rm) → sign ext → Rn,Rm + 1 → Rm	0110nnnnmmmm0100	-	1	MOV.B @R0,R1
MOV.W @Rm+,Rn	Move Data		(Rm) → sign ext → Rn, Rm + 2 → Rm	0110nnnnmmmm0101	-	1	
MOV.L @Rm+,Rn	Move Data		(Rm) → Rn, Rm + 4 → Rm	0110nnnnmmmm0110	-	1	MOV.L @R0+,R1
MOV.B Rm,@(R0,Rn)	Move Data		Rm → (R0 + Rn)	0000nnnnmmmm0100	-	1	MOV.B R1,@(R0,R2)
MOV.W Rm,@(R0,Rn)	Move Data		Rm → (R0 + Rn)	0000nnnnmmmm0101	-	1	
MOV.L Rm,@(R0,Rn)	Move Data		Rm → (R0 + Rn)	0000nnnnmmmm0110	-	1	
MOV.B @(R0,Rm),Rn	Move Data		(R0 + Rm) → sign extension → Rn	0000nnnnmmmm1100	-	1	
MOV.W @(R0,Rm),Rn	Move Data		(R0 + Rm) → sign extension → Rn	0000nnnnmmmm1101	-	1	MOV.W @(R0,R2),R1
MOV.L @(R0,Rm),Rn	Move Data		(R0 + Rm) → Rn	0000nnnnmmmm1110	-	1	
MOV #imm,Rn	Move Immediate Data	Stores immediate data, sign-extended to a longword, into general register Rn.	imm → sign extension → Rn	1110nnnniiiiiiii	-	1	MOV #H'80,R1
MOV.W @(disp,PC),Rn	Move Immediate Data	Stores immediate data, sign-extended to a longword, into general register Rn.	(disp × 2 + PC) → sign ext → Rn	1001nnnndddddddd	-	1	MOV.W IMM,R2

MOV.L @(disp,PC),Rn	Move Immediate Data	Stores immediate data, sign-extended to a longword, into general register Rn.	(disp × 4 + PC) → Rn	1101nnnndddddddd	-	1	MOV.L @(4,PC),R3
MOV.B @(disp,GBR),R0	Move Peripheral Data	Transfers the source operand to the dest. Designed for the peripheral module area.	(disp + GBR) → sign ext → R0	11000100dddddddd	-	1	
MOV.W @(disp,GBR),R0	Move Peripheral Data	Transfers the source operand to the dest. Designed for the peripheral module area.	(disp × 2 + GBR) → sign ext → R0	11000101dddddddd	-	1	
MOV.L @(disp,GBR),R0	Move Peripheral Data	Transfers the source operand to the dest. Designed for the peripheral module area.	(disp × 4 + GBR) → R0	11000110dddddddd	-	1	MOV.L @(2,GBR),R0
MOV.B R0,@(disp,GBR)	Move Peripheral Data	Transfers the source operand to the dest. Designed for the peripheral module area.	R0 → (disp + GBR)	11000000dddddddd	-	1	MOV.B R0,@(1,GBR)
MOV.W R0,@(disp,GBR)	Move Peripheral Data	Transfers the source operand to the dest. Designed for the peripheral module area.	R0 → (disp × 2 + GBR)	11000001dddddddd	-	1	
MOV.L R0,@(disp,GBR)	Move Peripheral Data	Transfers the source operand to the dest. Designed for the peripheral module area.	R0 → (disp × 4 + GBR)	11000010dddddddd	-	1	
MOV.B R0,@(disp,Rn)	Move Structure Data	Transfers the source operand to the dest. Designed for structure or a stack.	R0 → (disp + Rn)	10000000nnnndddd	-	1	
MOV.W R0,@(disp,Rn)	Move Structure Data	Transfers the source operand to the dest. Designed for structure or a stack.	R0 → (disp × 2 + Rn)	10000001nnnndddd	-	1	
MOV.L Rm,@(disp,Rn)	Move Structure Data	Transfers the source operand to the dest. Designed for structure or a stack.	Rm → (disp × 4 + Rn)	0001nnnnmmmmdddd	-	1	MOV.L R0,@(H'F,R1)
MOV.B @(disp,Rm),R0	Move Structure Data	Transfers the source operand to the dest. Designed for structure or a stack.	(disp + Rm) → sign extension → R0	10000100mmmmdddd	-	1	
MOV.W @(disp,Rm),R0	Move Structure Data	Transfers the source operand to the dest. Designed for structure or a stack.	(disp × 2 + Rm) → sign extension → R0	10000101mmmmdddd	-	1	
MOV.L @(disp,Rm),Rn	Move Structure Data	Transfers the source operand to the dest. Designed for structure or a stack.	(disp × 4 + Rm) → Rn	0101nnnnmmmmdddd	-	1	MOV.L @(2,R0),R1
MOVA @(disp,PC),R0	Move Effective Address	Stores the effective address of the source operand into general register R0. The 8-bit displacement is zero-extended and quadrupled	disp × 4 + PC → R0	11000111dddddddd	-	1	MOVA @(0,PC),R0
MOVT Rn	Move T Bit	Stores the T bit value into general register Rn. When T = 1, 1 is stored in Rn, and when T = 0, 0 is stored in Rn.	T → Rn	0000nnnn00101001	-	1	MOVT R0
MUL.L Rm,Rn	Multiply Long	Performs 32-bit multiplication of the contents of general registers Rn and Rm, and stores the bottom 32 bits of the result in the MACL register. The MACH register data does not change.	Rn × Rm → MACL	0000nnnnmmmm0111	-	2-4	MULL R0,R1
MULS.W Rm,Rn	Multiply as Signed Word	Performs 16-bit multiplication of the contents of general registers Rn and Rm, and stores the 32-bit result in the MACL register. The operation is signed and the MACH register data does not change.	Signed operation, Rn × Rm → MACL	0010nnnnmmmm1111	-	1-3	MULS R0,R1
MULS Rm,Rn							
MULU.W Rm,Rn	Multiply as Unsigned Word	Performs 16-bit multiplication of the contents of general registers Rn and Rm, and stores the 32-bit result in the MACL register. The operation is unsigned and the MACH register data does not change.	Unsigned, Rn × Rm → MACL	0010nnnnmmmm1110	-	1-3	MULU R0,R1
MULU Rm,Rn							
NEG R0,R1	Negate	Takes the two's complement of data in general register Rm, and stores the result in Rn. This effectively subtracts Rm data from 0, and stores the result in Rn.	0 – Rm → Rn	0110nnnnmmmm1011	-	1	NEG R0,R1
NEGC Rm,Rn	Negate with Carry	Subtracts general register Rm data and the T bit from 0, and stores the result in Rn. If a borrow is generated, T bit changes accordingly. This instruction is used for inverting the sign of a value that has more than 32 bits.	0 – Rm – T → Rn, Borrow → T	0110nnnnmmmm1010	-	1	NEGC R1,R1
NOP	No operation	Increments the PC to execute the next instruction.	No operation	0000000000001001	-	1	NOP
NOT Rm,Rn	NOT—Logical Complement	Takes the one's complement of general register Rm data, and stores the result in Rn. This effectively inverts each bit of Rm data and stores the result in Rn.	~Rm → Rn	0110nnnnmmmm0111	-	1	NOT R0,R1
OR Rm,Rn	OR Logical	Logically ORs the contents of general registers Rn and Rm, and stores the result in Rn. The contents of general register R0 can also be ORed with zero-extended 8-bit immediate data, or 8-bit memory data accessed by using indirect indexed GBR addressing can be ORed with 8-bit immediate data.	Rn | Rm → Rn	0010nnnnmmmm1011	-	1	OR R0,R1
OR #imm,R0	OR Logical		R0 | imm → R0	11001011iiiiiiii	-	1	OR #H'F0,R0
OR.B #imm,@(R0,GBR)	OR Logical		(R0 + GBR) | imm → (R0 + GBR)	11001111iiiiiiii	-	3	OR.B #H'50,@(R0,GBR)
ROTCL Rn	Rotate with Carry Left	Rotates the contents of general register Rn and the T bit to the left by one bit, and stores the result in Rn. The bit that is shifted out is transferred to the T bit	T ← Rn ← T	0100nnnn00100100	MSB	1	ROTCL R0
ROTCR Rn	Rotate with Carry Right	Rotates the contents of general register Rn and the T bit to the right by one bit, and stores the result in Rn. The bit that is shifted out of is transferred to the T bit	T → Rn → T	0100nnnn00100101	LSB	1	ROTCR R0
ROTL Rn	Rotate Left	Rotates the contents of general register Rn to the left by one bit, and stores the result in Rn . The bit that is shifted out of the operand is transferred to the T bit.	T ← Rn ← MSB	0100nnnn00000100	MSB	1	ROTL R0
ROTR Rn	Rotate Right	Rotates the contents of general register Rn to the right by one bit, and stores the result in Rn . The bit that is shifted out of the operand is transferred to the T bit.	LSB → Rn → T	0100nnnn00000101	LSB	1	ROTR R0
RTE	Return from Exception	Returns from an interrupt routine. The PC and SR values are restored from the stack, and the program continues from the address specified by the restored PC value. The T bit is used as the LSB bit in the SR register restored from the stack area.	Delayed branch, Stack area → PC/SR	0000000000101011	LSB	4	RTE
RTS	Return from Subroutine	Returns from a subroutine procedure. The PC values are restored from the PR, and the program continues from the address specified by the restored PC value. This instruction is used to return to the program from a subroutine program called by a BSR, BSRF, or JSR instruction.	Delayed branch, PR → PC	0000000000001011	-	2	RTS
SETRC Rm	Set Repeat Count to RC	Sets the repeat count to the SR register's RC counter. When the operand is a register, the bottom 12 bits are used as the repeat count. When the operand is an immediate data value, 8 bits are used as the repeat count. Set repeat control flags to RF1, RF0 bits of the SR register. Use of the SETRC instruction is subject to any limitations.	Rm[11:0] RCCSR[27:16] Repeat control flag → RF1, RF0	0100mmmm00010100	-	1	
SETRC #imm			imm → RC [23:26] zeros → SR[27:24], Repeat control flag → RF1, RF0	10000010iiiiiiii	-	1	SETRC #32
SETT	Set T Bit	Sets the T bit to 1.	1 → T	0000000000011000	1	1	SETT
SHAL Rn	Shift Arithmetic Left	rithmetically shifts the contents of general register Rn to the left by one bit, and stores the result in Rn. The bit that is shifted out of the operand is transferred to the T bit	T ← Rn ← 0	0100nnnn00100000	MSB	1	SHAL R0
SHAR Rn	MSB → Rn → T	Arithmetically shifts the contents of general register Rn to the right by one bit, and stores the result in Rn. The bit that is shifted out is transferred to the T bit	MSB → Rn → T	0100nnnn00100001	LSB	1	SHAR R0
SHLL Rn	T ← Rn ← 0	Logically shifts the contents of general register Rn to the left by one bit, and stores the result in Rn. The bit that is shifted out of the operand is transferred to the T bit	T ← Rn ← 0	0100nnnn00000000	MSB	1	SHLL R0
SHLL2	Shift Logical Left n Bits	Logically shifts the contents of general register Rn to the left by 2, 8, or 16 bits, and stores the result in Rn. Bits that are shifted out of the operand are not stored	Rn << 2 → Rn	0100nnnn00001000	-	1	SHLL2 R0
SHLL8	Shift Logical Left n Bits		Rn << 8 → Rn	0100nnnn00011000	-	1	SHLL8 R0
SHLL16 Rn	Shift Logical Left n Bits		Rn << 16 → Rn	0100nnnn00101000	-	1	SHLL16 R0
SHLR Rn	Shift Logical Right	Logically shifts the contents of general register Rn to the right by one bit, and stores the result in Rn. The bit that is shifted out of the operand is transferred to the T bit	0 → Rn → T	0100nnnn00000001	LSB	1	SHLR R0
SHLR2 Rn	Shift Logical Right n Bits	Logically shifts the contents of general register Rn to the right by 2, 8, or 16 bits, and stores the result in Rn. Bits that are shifted out of the operand are not stored	Rn>>2 → Rn	0100nnnn00001001	-	1	SHLR2 R0
SHLR8 Rn	Shift Logical Right n Bits		Rn>>8 → Rn	0100nnnn00011001	-	1	SHLR8 R0
SHLR16 Rn	Shift Logical Right n Bits		Rn>>16 → Rn	0100nnnn00101001	-	1	SHLR16 R0
SLEEP	Sleep	Sets the CPU into power-down mode. CPU waits for an interrupt request.	Sleep	0000000000011011	-	3	SLEEP
STC SR,Rn	Store Control Register	Stores control register into a specified destination.	SR → Rn	0000nnnn00000010	-	1	STC SR,R0
STC GBR,Rn	Store Control Register	Stores control register into a specified destination.	GBR → Rn	0000nnnn00010010	-	1	
STC VBR,Rn	Store Control Register	Stores control register into a specified destination.	VBR → Rn	0000nnnn00100010	-	1	
STC MOD,Rn	Store Control Register	Stores control register into a specified destination.	MOD → Rn	0000nnnn01010010	-	1	
STC RE,Rn	Store Control Register	Stores control register into a specified destination.	RE → Rn	0000nnnn01110010	-	1	
STC RS,Rn	Store Control Register	Stores control register into a specified destination.	RS → Rn	0000nnnn01100010	-	1	
STC.L SR,@-Rn	Store Control Register	Stores control register into a specified destination.	Rn – 4 → Rn, SR → (Rn)	0100nnnn00000011	-	2	
STC.L GBR,@-Rn	Store Control Register	Stores control register into a specified destination.	Rn – 4 → Rn, GBR → (Rn)	0100nnnn00010011	-	2	STC.L GBR,@-R15
STC.L VBR,@-Rn	Store Control Register	Stores control register into a specified destination.	Rn – 4 → Rn, VBR → (Rn)	0100nnnn00100011	-	2	
STC.L MOD,@-Rn	Store Control Register	Stores control register into a specified destination.	Rn – 4 → Rn,MOD → (Rn)	0100nnnn01010011	-	2	
STC.L RE,@-Rn	Store Control Register	Stores control register into a specified destination.	Rn – 4 → Rn, RE → (Rn)	0100nnnn01110011	-	2	
STC.L RS,@-Rn	Store Control Register	Stores control register into a specified destination.	Rn – 4 → Rn, RS → (Rn)	0100nnnn01100011	-	2	
STS MACH,Rn	Store System Register	Stores data from system register into a specified destination	MACH → Rn	0000nnnn00001010	-	1	STS MACH,R0
STS MACL,Rn	Store System Register	Stores data from system register into a specified destination	MACL → Rn	0000nnnn00011010	-	1	
STS PR,Rn	Store System Register	Stores data from system register into a specified destination	PR → Rn	0000nnnn00101010	-	1	
STS DSR,Rn	Store System Register	Stores data from system register into a specified destination	DSR → Rn	0000nnnn01101010	-	1	
STS A0,Rn	Store System Register	Stores data from system register into a specified destination	A0 → Rn	0000nnnn01111010	-	1	
STS X0,Rn	Store System Register	Stores data from system register into a specified destination	X0→Rn	0000nnnn10001010	-	1	
STS X1,Rn	Store System Register	Stores data from system register into a specified destination	X1→Rn	0000nnnn10011010	-	1	
STS Y0,Rn	Store System Register	Stores data from system register into a specified destination	Y0→Rn	0000nnnn10101010	-	1	
STS Y1,Rn	Store System Register	Stores data from system register into a specified destination	Y1→Rn	0000nnnn10111010	-	1	
STS.L MACH,@–Rn	Store System Register	Stores data from system register into a specified destination	Rn – 4 → Rn,MACH → (Rn)	0100nnnn00000010	-	1	
STS.L MACL,@–Rn	Store System Register	Stores data from system register into a specified destination	Rn – 4 → Rn,MACL → (Rn)	0100nnnn00010010	-	1	
STS.L PR,@–Rn	Store System Register	Stores data from system register into a specified destination	Rn – 4 → Rn,PR → (Rn)	0100nnnn00100010	-	1	STS.L PR,@–R15
STS.L DSR,@–Rn	Store System Register	Stores data from system register into a specified destination	Rn – 4 → Rn,DSR → (Rn)	0100nnnn01100010	-	1	
STS.L A0,@–Rn	Store System Register	Stores data from system register into a specified destination	Rn – 4 → Rn, A0 → (Rn)	0100nnnn01100010	-	1	
STS.L X0,@-Rn	Store System Register	Stores data from system register into a specified destination	Rn–4→Rn,X0→(Rn)	0100nnnn10000010	-	1	
STS.L X1,@-Rn	Store System Register	Stores data from system register into a specified destination	Rn–4→Rn,X1→(Rn)	0100nnnn10010010	-	1	
STS.L Y0,@-Rn	Store System Register	Stores data from system register into a specified destination	Rn–4→Rn,Y0→(Rn)	0100nnnn10100010	-	1	
STS.L Y1,@-Rn	Store System Register	Stores data from system register into a specified destination	Rn–4→Rn,Y1→(Rn)	0100nnnn10110010	-	1	
SUB Rm,Rn	Subtract Binary	Subtracts general register Rm data from Rn data, and stores the result in Rn. To subtract immediate data, use ADD #imm,Rn.	Rn – Rm → Rn	0011nnnnmmmm1000	-	1	SUB R0,R1
SUBC Rm,Rn	Subtract with Carry	Subtracts Rm data and the T bit value from general register Rn data, and stores the result in Rn. The T bit changes according to the result. This instruction is used for subtraction of data that has more than 32 bits.	Rn – Rm– T → Rn, Borrow → T	0011nnnnmmmm1010	-	1	SUBC R3,R1
SUBV Rm,Rn	Subtract with V Flag Underflow Check	Subtracts Rm data from general register Rn data, and stores the result in Rn. If an underflow occurs, the T bit is set to 1.	Rn – Rm → Rn, underflow → T	0011nnnnmmmm1011	Under Flow	1	SUBV R0,R1
SWAP.B Rm,Rn	Swap Register Halves	Swaps the upper and lower bytes of the general register Rm data, and stores the result in Rn. If a byte is specified, bits 0 to 7 of Rm are swapped for bits 8 to 15. The upper 16 bits of Rm are transferred to the upper 16 bits of Rn. If a word is specified, bits 0 to 15 of Rm are swapped for bits 16 to 31.	Rm → Swap upper and lower halves of lower 2 bytes → Rn	0110nnnnmmmm1000	-	1	SWAP.B R0,R1
SWAP.W Rm,Rn	Swap Register Halves		Rm → Swap upper and lower word → Rn	0110nnnnmmmm1001	-	1	SWAP.W R0,R1
TAS.B @Rn	Test and Set	Reads byte data from the address specified by general register Rn, and sets the T bit to 1 if the data is 0, or clears the T bit to 0 if the data is not 0. Then, data bit 7 is set to 1, and the data is written to the address specified by Rn. During this operation, the bus is not released.	When (Rn) is 0, 1 → T, 1 → MSB of (Rn)	0100nnnn00011011	reslt	4	TAS.B @R7
TRAPA #imm	Trap Always	Starts the trap exception processing. The PC and SR values are stored on the stack, and the program branches to an address specified by the vector. The vector is a memory address obtained by zero-extending the 8-bit immediate *4 . The PC is the start address of the next instruction. TRAPA and RTE are used for system calls.	PC/SR → Stack area, (imm × 4 + VBR) → PC	11000011iiiiiiii	-	8	TRAPA #H'20
TST Rm,Rn	Test Logical	Logically ANDs the contents of general registers Rn and Rm, and sets the T bit to 1 if the result is 0 or clears the T bit to 0 if the result is not 0. The Rn data does not change. The contents of general register R0 can also be ANDed with zero-extended 8-bit immediate data, or the contents of 8-bit memory accessed by indirect indexed GBR addressing can be ANDed with 8- bit immediate data. The R0 and memory data do not change.	Rn & Rm, when result is 0, 1 → T	0010nnnnmmmm1000	reslt	1	TST R0,R0
TST #imm,R0	Test Logical		R0 & imm, when result is 0, 1 → T	11001000iiiiiiii	reslt	1	TST #H'80,R0
TST.B #imm, @(R0,GBR)	Test Logical		(R0 + GBR) & imm, when result is 0, 1 → T	11001100iiiiiiii	reslt	1	TST.B #H'A5,@(R0,GBR)
XOR Rm,Rn	Exclusive OR Logical	Exclusive ORs the contents of general registers Rn and Rm, and stores the result in Rn. The contents of general register R0 can also be exclusive ORed with zero-extended 8-bit immediate data, or 8-bit memory accessed by indirect indexed GBR addressing can be exclusive ORed with 8-bit immediate data.	Rn ^ Rm → Rn	0010nnnnmmmm1010	-	1	XOR R0,R1
XOR #imm,R0	Exclusive OR Logical		R0 ^ imm → R0	11001010iiiiiiii	-	1	XOR #H'F0,R0
XOR.B #imm,@(R0,GBR)	Exclusive OR Logical		(R0 + GBR) ^ imm → (R0 + GBR)	11001110iiiiiiii	-	3	XOR.B #H'A5,@(R0,GBR)
XTRCT Rm,Rn	Extract	Extracts the middle 32 bits from the 64 bits of coupled general registers Rm and Rn, and stores the 32 bits in Rn	Rm: Center 32 bits of Rn → Rn	0010nnnnmmmm1101	-	1	XTRCT R0,R1

IBM 370

Instruction	Description	Format	Opcode	Class
A R1,D2(X2,B2)	Add	RX	5A	c
AD R1,D2(X2,B2)	Add Normalized (L)	RX	6A	c
ADR R1,R2	Add Normalized (L)	RR	2A	c
AE R1,D2(X2,B2)	Add Normalized (S)	RX	7A	c
AER R1,R2	Add Normalized (S)	RR	3A	c
AH R1,D2(X2,B2)	Add Halfword	RX	4A	c
AL R1,D2(X2,B2)	Add Logical	RX	5E	c
ALR R1,R2	Add Logical	RR	1E	c
AP D1(L1,B1),D2(L2,B2)	Add Decimal	SS	FA	c
AR R1,R2	Add	RR	1A	c
AU R1,D2(X2.B2)	Add Unnormalized (S)	RX	7E	c
AUR R1,R2	Add Unnormalized (S)	RR	3E	c
AW R1,D2(X2,B2)	Add Unnormalized (L)	RX	6E	c
AWR R1,R2	Add Unnormalized (L)	RR	2E	c
AXR R1,R2	Add Normalized (E)	RR	36	c
BAL R1,D2(X1,B2)	Branch and Link	RX	45	
BALR R1,R2	Branch and Link	RR	05	
BAS R1,D2(X2.B2)	Branch and Save	RX	4D	
BASR R1,R2	Branch and Save	RR	0D	
BC M1,D2(X2.B2)	Branch on Condition	RS	47	
BCR M1,R2	Branch on Condition	RR	07	
BCT R1,D2(X2.B2)	Branch on Count	RX	46	
BCTR R1,R2	Branch on Count	RR	06	
BXH R1,R3,D2(B2)	Branch on Index High	RS	86	
BXLE R1,R3,D2(B2)	Branch on Index Low	RS	87	
C R1,D2(X2,.B2)	Compare	RX	59	c
CD R1,D2(X2,B2)	Compare (L)	RX	69	c
CDR R1,R2	Compare (L)	RR	29	c
CDS R1,R3,D2(B2)	Compare Double and Swap	RS	BB	c
CE R1,D2(X2,B2)	Compare (S)	RX	79	c
CER R1,R2	Compare (S)	RR	39	c
CH R1,D2(X2,B2)	Compare Halfword	RX	49	c
CL R1,D2(X2,B2)	Compare Logical	RX	55	c
CLC D1,(L,B1),D2(B2)	Compare Logical	SS	D5	c
CLCL R1,R2	Compare Logical Long	RR	0F	i c
CLI D1(B1),I2	Compare Logical	SI	95	c
CLM R1,M3,D2(B2)	Comp Logical Chars under Mask	RS	BD	c
CLR R1,R2	Compare Logical	RR	15	c
CLRCH D2(B2)	Clear Channel	S	9F01	pc
CLRIO D2(B2)	Clear I/O	S	9D01	pc
CONCS D2(B2)	Connect Channel Set	S	B200	pc
CP D1(L1,B1),D2(L2,B2)	Compare Decimal	SS	F9	c
CR R1,R2	Compare	RR	19	c
CS R1,R3,D2,(B2)	Compare and Swap	RS	BA	c
CVB R1,D2(X2.B2)	Convert to Binary	RX	4F	
CVD R1,D2(X2,B2)	Convert to Decimal	RX	4E	
D R1,D2(X2,B2)	Divide	RX	5D	
DD R1,D2(X2,B2)	Divide (L)	RX	5D	
DDR R1,R2	Divide (L)	RR	2D	
DE R1,D2(X2,B2)	Divide (S)	RX	7D	
DER R1,R2	Divide (S)	RR	3D	
DISCS D2(B2)	Disconnect Channel Set	S	B201	pc
DP D1(L1,B1),D2(L2,B2)	Divide Decimal	SS	FD	
DR R1,R2	Divide	RR	1D	
ED D1,LI,B1),D2(B2)	Edit	SS	DE	c
EDMK D1(L,B1),D2(B2)	Edit and Mark	SS	DF	c
EPAR R1	Extract Primary ASN	RRE	B226	q
ESAR R1	Extract Secondary ASN	RRE	B227	q
EX R1,D2(X2,B2)	Execute	RX	44	
HDR R1,R2	Halve (L)	RR	24	
HDV D2(B2)	Halt Device	S	9E01	pc
HER R1,R2	Halve (S)	RR	34	
HIO D2(B2)	Halt I/O	S	9E00	pc
IAC R1	Insert Address Space Control	RRE	B224	qp
IC R1,D2(X2,B2)	Insert Character	RX	43	
ICM R1,M3,D2(B2)	Insert Characters under Mask	RS	BS	c
IPK	Insert PSW Key	S	B20B	q
IPTE R1,R2	Invalidate Page Table Entry	RRE	B221	p
ISK R1,R2	Insert Storage Key	RR	09	p
ISKE R1,R2	Insert Storage Key Extended	RRE	B229	p
IVSK R1,R2	Insert Virtual Storage Key	RRE	B223	q
L R1,D2(X2,B2)	Load	RX	58	
LA R1,D2(X2,B2)	Load Address	RX	41	
LASP D1(B1),D2(B2)	Load Address Space Parameters	SSE	E500	pc
LCDR R1,R2	Load Complement (L)	RR	23	c
LCER R1,R2	Load Complement (S)	RR	33	c
LCR R1,R2	Load Complement	RR	13	c
LCTL R1,R3,D2(B2)	Load Control	RS	B7	p
LD R1,D2(X2,B2)	Load (L)	RX	68	
LDR R1,R2	Load (L)	RR	28	
LE R1,D2(X2,B2)	Load (S)	RX	78	
LER R1,R2	Load (S)	RR	38	
LH R1,D2(X2,B2)	Load Halfword	RX	48	
LM R1,R2,D2(B2)	Load Multiple	RS	98	
LNDR R1,R2	Load Negative (L)	RR	21	c
LNER R1,R2	Load Negative (S)	RR	31	c
LNR R1,R2	Load Negative	RR	11	c
LPDR R1,R2	Load Positive (L)	RR	20	c
LPER R1,R2	Load Positive (S)	RR	30	c
LPR R1,R2	Load Positive	RR	10	c
LPSW D2(B2)	Load PSW	S	82	pn
LR R1,R2	Load	RR	18	
LRA R1,D2(X2,B2)	Load Real Address	RX	B1	pc
LRDR R1,R2	Load Rounded (E/L)	RR	25	
LRER R1,R2	Load Rounded (L/S)	RR	35	
LTDR R1,R2	Load and Test (L)	RR	22	c
LTER R1,R2	Load and Test (S)	RR	32	c
LTR R1,R2	Load and Test	RR	12	c
M R1,D2(X2,B2)	Multiply	RX	5C	
MC D1(B1),I2	Monitor Call	SI	AF	
MD R1,D2(X2,B2)	Multiply (L)	RX	6C	
MDR R1,R2	Multiply (L)	RR	2C	
ME R1,D2(X2,B2)	Multiply (S/L)	RX	7C	
MER R1,R2	Multiply (S/L)	RR	3C	
MH R1,D2(X2,B2)	Multiply Halfword	RX	4C	
MP D1(L1,B1),D2(L2,B2)	Multiply Decimal	SS	FC	
MR R1,R2	Multiply	RR	1C	
MVC D1(L,B1),D2(B2)	Move	SS	D2	
MVCIN D1(L,B1),D2(B2)	Move Inverse	SS	E8	
MVCK D1(R1,B1),D2(B2),R3	Move with Key	SS	D9	qc
MVCL R1,R2	Move Long	RR	0E	i c
MVCP D1(R1,B1),D2(B2),R3	Move to Primary	SS	DA	qc
MVCS D1(R1,B1),D2(B2),R3	Move to Secondary	SS	DB	qc
MVI D1(B1),L2	Move	SI	92	
MVN D1(L,B1),D2(B2)	Move Numerics	SS	D1	
MVO D1(L1,B1),D2(L2,B2)	Move with Offset	SS	F1	
MVZ D1(L,B1),D2(B2)	Move Zones	SS	D3	
MXD R1,D2(X2,B2)	Multiply (L/E)	RX	67	
MXDR R1,R2	Multiply (L/E)	RR	27	
MXR R1,R2	Multiply (E)	RR	26	
N R1,D2(X2,B2)	AND	RX	54	c
NC D1(L,B1),D2(B2)	AND	SS	D4	c
NI D1(B1),I2	AND	SI	94	c
NR R1,R2	AND	RR	14	c
O R1,D2(X2,B2)	OR	RX	56	c
OC D1(L,B1),D2(B2)	OR	SS	D6	c
OI D1(B1I.I2	OR	SI	96	c
OR R1,R2	OR	RR	16	c
PACK D1(L1,B1),D2(L2,B2)	Pack	SS	F2	
PC D2(B2)	Program Call	S	B218	q
PT R1,R2	Program Transfer	RRE	B228	q
PTLB	Purge TLB	S	B20D	p
RDD D1(B1).I2	Read Direct	SI	85	p
RIO D2(B2)	Resume I/O	S	9C02	pc
RRB D2(B2)	Reset Reference Bit	S	B213	pc
RABE R1,R2	Reset Reference Bit Extended	RRE	B22A	pc
S R1,D2(X2,B2)	Subtract	RX	5B	c
SAC D2(B2)	Set Address Space Control	S	B219	q
SCK D2(B2)	Set Clock	S	B204	pc
SCKC D2(B2)	Set Clock Comparator	S	B206	p
SD R1,D2(X2,B2)	Subtract Normalized (L)	RX	6B	c
SDR R1,R2	Subtract Normalized (L)	RR	2B	c
SE R1,D2(X2,B2)	Subtract Normalized (S)	RX	7B	c
SER R1,R2	Subtract Normalized (S)	RR	3B	c
SH R1,D2(X2,B2)	Subtract Halfword	RX	4B	c
SIGP R1,R3,D2(B2)	Signal Processor	RS	AE	pc
SIO D2(B2)	Start I/O	S	9C00	pc
SIOF D2(B2)	Start I/O Fast Release	S	9C01	pc
SL R1,D2(X2,B2)	Subtract Logical	RX	5F	c
SLA R1,D2(B2)	Shift Left Single	RS	8B	c
SLDA R1,D2(B2)	Shift Left Double	RS	8F	c
SLDL R1,D2(B2)	Shift Left Double Logical	RS	8D	
SLL R1,D2(B2)	Shift Left Single Logical	RS	89	
SLR R1,R2	Subtract Logical	RR	1F	c
SP D1(L1,B1),D2(L2,B2)	Subtract Decimal	SS	FB	c
SPKA D2(B2)	Set PSW Key from Address	S	B20A	q
SPM R1	Set Program Mask	RR	04	n
SPT D2,(B2)	Set CPU Timer	S	B208	p
SPX D2,(B2)	Set Prefix	S	B210	p
SR R1,R2	Subtract	RR	1B	c
SRA R1,D2(B2)	Shift Right Single	RS	8A	c
SRDA R1,D2(B2)	Shift Right Double	RS	8E	c
SRDL R1,D2(B2)	Shift Right Double Logical	RS	8C	
SRL R1,D2(B2)	Shift Right Single Logical	RS	88	
SRP D1(L1.B1),D2(B2),I3	Shift and Round Decimal	SS	F0	c
SSAR R1	Set Secondary ASN	RRE	B225	q
SSK R1,R2	Set Storage Key	RR	08	p
SSKE R1,R2	Set Storage Key Extended	RRE	B22B	p
SSM D2,(B2)	Set System Mask	S	80	p
ST R1,D2(X2,B2)	Store	RX	50	
STAP D2,(B2)	Store CPU Address	S	B212	p
STC R1,D2(X2,B2)	Store Character	RX	42	
STCK D2,(B2)	Store Clock	S	B205	c
STCKC D2,(B2)	Store Clock Comparator	S	B207	p
STCM R1,M3,D2(B2)	Store Characters under Mask	RS	BE	
STCTL R1,R3,D2(B2)	Store Control	TS	B6	p
STD R1,D2(X2,B2)	Store (L)	RX	60	
STE R1,D2(X2,B2)	Store (S)	RX	70	
STH R1,D2(X2,B2)	Store Halfword	RX	40	
STIDC D2,(B2)	Store Channel ID	S	B203	pc
STIDP D2,(B2)	Store CPU ID	S	B202	p
STM R1,R3,D2(B2)	Store Multiple	RS	90	
STNSM D1(B1),I2	Store Then AND System Mask	SI	AC	p
STOSM D1(B1),I2	Store Then OR System Mask	SI	AD	p
STPT D2,(B2)	Store CPU Timer	S	B209	p
STPX D2,(B2)	Store Prefix	S	B211	p
SU R1,D2(X2,B2)	Subtract Unnormalized (S)	RX	7F	
SUR R1,R2	Subtract Unnormalized (S)	RR	3F	
SVC I	Supervisor Call	RR	0A	
SW R1,D2(X2,B2)	Subtract Unnormalized (L)	RX	6F	
SWR R1,R2	Subtract Unnormalized (L)	RR	2F	
SXR R1,R2	Subtract Normalized (E)	RR	37	
TB R1,R2	Test Block	RRE	B22C	ipc
TCH D2(B2)	Test Channel	S	9F00	pc
TIO D2(B2)	Test I/O	S	9D00	pc
TM D1(B1),I2	Test under Mask	SI	91	c
TPROT D1(B1),D2(B2)	Test Protection	SSE	E501	pc
TR D1(L,B1),D2(B2)	Translate	SS	DC	
TRT D1(L,B1),D2(B2)	Translate and Test	SS	DD	c
TS D2(B2)	Test and Set	S	93	c
UNPK D1(L1,B1),D2(L2,B2)	Unpack	SS	F3	
VA VR1, VR3,RS2(RT2)	Add	VST	A420	IM
VACD VR1,RS2(RT2)	Accumulate (L)	VST	A417	IM
VACDR VR1,VR2	Accumulate (L)	VV	A517	IM
VACE VR1,RS2(RT2)	Accumulate (S/L)	VST	A407	IM
VACER VR1,VR2	Accumulate (S/L)	VV	A507	IM
VACRS D2(B2)	Restore VAC	S	A6CB	NO P
VACSV D2(B2)	Save VAC	S	A6CA	NO P
VAD VR1,VR3,RS2(RT2)	Add (L)	VST	A410	IM
VADQ VR1,FR3,VR2	Add (L)	QV	A490	IM
VADR VR1,VR3,VR2	Add (L)	VV	A510	IM
VADS VR1,FR3,RS2(RT2)	Add (L)	QST	A490	IM
VAE VR1,VR3,RS2(RT2)	Add (S)	VST	A400	IM
VAEQ VR1,FR3,VR2	Add (S)	QV	A580	IM
VAER VR1, VR3,VR2	Add (S)	VV	A500	IM
VAES VR1,FR3,RS2(RT2)	Add (S)	QST	A480	IM
VAQ VR1,GR3,VR2	Add	QV	A5A0	IM
VAR VR1,VR3,VR2	Add	VV	A520	IM
VAS VR1,GR3,RS2(RT2)	Add	QST	A4A0	IM
VC M1,VR3,RS2(RT2)	Compare	VST	A428	IC
VCD M1,VR3,RS2(RT2)	Compare (L)	VST	A418	IC
VCDQ M1,FR3,VR2	Compare (L)	QV	A598	IC
VCDR M1,VR3,VR2	Compare (L)	VV	A518	IC
VCDS M1,FR3,RS2(RT2)	Compare (L)	QST	A498	IC
VCE M1,VR3,RS2(RT2)	Compare (S)	VST	A408	IC
VCEQ M1,FR3,VR2	Compare (S)	QV	A588	IC
VCER M1,VR3,VR2	Compare (S)	VV	A508	IC
VCES M1,FR3,RS2(RT2)	Compare (S)	QST	A488	IC
VCOVM GR1	Count Ones in VMR	RRE	A643	NC C
VCQ M1,GR3,VR2	Compare	QV	A5A8	IC
VCR M1.VR3,VR2	Compare	VV	A528	IC
VCS M1,GR3,RS2(RT2)	Compare	QST	A4A8	IC
VCVM	Complement VMR	RRE	A641	NC
VCZVM GR1	Count Left Zeros in VMR	RRE	A642	NC C
VDD VR1, VR3,RS2(RT2)	Divide (L)	VST	A413	IM
VDDQ VR1,FR3,VR2	Divide (L)	QV	A493	IM
VDDR VR1,VR3,VR2	Divide (L)	VV	A513	IM
VDDS VR1,FR3,RS2(RT2)	Divide (L)	QST	A493	IM
VDE VR1,VR3,RS2(RT2)	Divide (S)	VST	A403	IM
VDEQ VR1,FR3,VR2	Divide (S)	QV	A583	IM
VDER VR1,VR3,VR2	Divide (S)	VV	A503	IM
VDES VR1,FR3,RS2(RT2)	Divide (S)	QST	A483	IM
VL VR1,RS2(RT2)	Load VST	VST	A409	IC
VLBIX VR1,GR3,D2(B2)	Load Bit Index	RSE	E428	IG C
VLCDR VR1,VR2	Load Complement (L)	VV	A552	IM
VLCER VR1,VR2	Load Complement (S)	VV	A542	IM
VLCR VR1,VR2	Load Complement	VV	A562	IM
VLCVM RS2	Load VMR Complement	VS	A681	NC
VLD VR1,RS2(RT2)	Load (L)	VST	A419	IC
VLDO VR1,FR2	Load (L)	QV	A599	IC
VLDR VR1,VR2	Load (L)	VV	A519	IC
VLE VR1,RS2(RT2)	Load (S)	VST	A409	IC
VLEL VR1,GR3,GR2	Load Element	VR	A628	N1
VLELD VR1,FR3,GR2	Load Element (L)	VR	A618	N1
VLELE VR1,FR3,GR2	Load Element (S)	VR	S608	N1
VLEQ VR1.FR2	Load (S)	QV	A608	IC
VLER VR1,VR2	Load (S)	VV	A589	IC
VLH VR1,RS2(RT2)	Load Halfword	VST	A429	IC
VLI VR1, VR3,D2(B2)	Load Indirect	RSE	E400	IC
VLID VR1,VR3,D2(B2)	Load Indirect (L)	RSE	E410	IC
VLIE VR1, VR3,D2(B2)	Load Indirect (S)	RSE	E400	IC
VLINT VR1,RS2(RT2)	Load Integer Vector	VST	A42A	IC
VLM VR1,RS2(RT2)	Load Matched	VST	A40A	IC
VLMD VR1,RS2(RT2)	Load Matched (L)	VST	A41A	IC
VLMDQ VR1,FR2	Load Matched (L)	QV	A59A	IC
VLMDR VR1,VR2	Load Matched (L)	VV	A51A	IC
VLME VR1,RS2(RT2)	Load Matched (S)	VST	A40A	IC
VLMEQ VR1,FR2	Load Matched (S)	QV	A58A	IC
VLMER VR1,VR2	Load Matched (S)	VV	A50A	IC
VLMQ VR1,GR2	Load Matched	QV	A5AA	IC
VLMR VR1,VR2	Load Matched	VV	A50A	IC
VLNDR VR1,VR2	Load Negative (L)	VV	A551	IM
VLNER VR1,VR2	Load Negative (S)	VV	A541	IM
VLNR VR1,VR2	Load Negative	VV	A561	IM
VLPDR VR1,VR2	Load Positive (L)	VV	A550	IM
VLPER VR1,VR2	Load Positive (S)	VV	A540	IM
VLPR VR1,VR2	Load Positive	VV	A560	IM
VLQ VR1,GR2	Load	QV	A5A9	IC
VLR VR1,VR2	Load	VV	A509	IC
VLVCA D2(B2)	Load VCT from Address	S	A6C4	NO C
VLVCU GR1	Load VCT and Update	RRE	A645	NO C
VLVM RS2	Load VMR	VS	A680	NC
VLY VR1,RS2(RT2)	Load Expanded	VST	A40B	IC
VLYD VR1,RS2(RT2)	Load Expanded (L)	VST	A41B	IC
VLYE VR1,RS2(RT2)	Load Expanded (S)	VST	A40B	IC
VLZDR VR1	Load Zero (L)	VV	A51B	IC
VLZER VR1	Load Zero (S)	VV	A50B	IC
VLZR VR1	Load Zero	VV	A50B	IC
VM VR1,VR3,RS2(RT2)	Multiply	VST	A522	IM
VMAD VR1,VR3,RS2(RT2)	Multiply and Add (L)	VST	A414	IM
VMADQ VR1,FR3,VR2	Multiply and Add (L)	QV	A594	IM
VMADS VR1,FR3,RS2(RT2)	Multiply and Add (L)	OST	A494	IM
VMAE VR1,VR3,RS2(RT2)	Multiply and Add (S/L)	VST	A404	IM
VMAEQ VR1,FR3,VR2	Multiply and Add (S/L)	QV	A584	IM
VMAES VR1,FR3,RS2(RT2)	Multiply and Add (S/L)	QST	A484	IM
VMCD VR1,VR3,RS2(RT2)	Multiply and Accumulate (L)	VST	A416	IM
VMCDR VR1,VR3,VR2	Multiply and Accumulate (L)	VV	A516	IM
VMCE VR1,VR3,RS2(RT2)	Multiply and Accumulate (S/L)	VST	A406	IM
VMCER VR1,VR3,VR2	Multiply and Accumulate (S/L)	VV	A506	IM
VMD VR1,VR3,RS2(RT2)	Multiply (L)	VST	A412	IM
VMDQ VR1,FR3,VR2	Multiply (L)	QV	A592	IM
VMDR VR1,VR3,VR2	Multiply (L)	VV	A512	IM
VMDS VR1,FR3,RS2(RT2)	Multiply (L)	QST	A492	IM
VME VR1,VR3,RS2(RT2)	Multiply (S/L)	VST	A402	IM
VMEQ VR1,FR3,VR2	Multiply (S/L)	QV	A582	IM
VMER VR1,VR3,VR2	Multiply (S/L)	VV	A502	IM
VMES VR1,FR3,RS2(RT2)	Multiply (S/L)	QST	A482	IM
VMNSD VR1,FR3,GR2	Minimum Signed (L)	VR	A611	IM
VMNSE VR1,FR3,GR2	Minimum Signed (S)	VR	A601	IM
VMQ VR1,GR3,VR2	Multiply	QV	A5A2	IM
VMR VR1,VR3,VR2	Multiply	VV	A522	IM
VMRRS D2(B2)	Restore VMR	S	A6C3	NZ
VMRSV D2(B2)	Save VMR	S	A6C1	NZ
VMS VR1,GR3,RS2(RT2)	Multiply	QST	A5A2	IM
VMSD VR1,VR3,RS2(RT2)	Multiply and Subtract (L)	VST	A415	IM
VMSDQ VR1,FR3,VR2	Multiply and Subtract (L)	QV	A595	IM
VMSDS VR1,FR3,RS2(RT2)	Multiply and Subtract (L)	QST	A495	IM
VMSE VR1,VR3,RS2(RT2)	Multiply and Subtract (S/L)	VST	A405	IM
VMSEQ VR1,FR3,VR2	Multiply and Subtract (S/L)	QV	A585	IM
VMSES VR1,FR3,RS2(RT2)	Multiply and Subtract (S/L)	QST	A485	IM
VMXAD VR1,FR3,GR2	Maximum Absolute (L)	VR	A612	IM
VMXAE VR1,FR3,GR2	Maximum Absolute (S)	VR	A602	IM
VMXSD VR1,FR3.GR2	Maximum Signed (L)	VR	A610	IM
VMXSE VR1,FR3,GR2	Maximum Signed (S)	VR	A600	IM
VN VR1,VR3,RS2(RT2)	AND	VST	A424	IM
VNQ VR1,GR3,VR2	AND	QV	A5A4	IM
VNR VR1,VR3,VR2	AND	VV	A524	IM
VNS VR1,GR3,RS2(RT2)	AND	QST	A4A4	IM
VNVM RS2	AND to VMR	VS	A684	NC
VO VR1,VR3,RS2(RT2)	OR	VST	A425	IM
BOQ VR1,GR3,VR2	OR	QV	A5A5	IM
VOR VR1,VR3,VR2	OR	VV	A525	IM
VOS VR1,GR3,RS2(RT2)	OR	QST	A5A5	IM
VOVM RS2	OR to VMR	VS	A685	NC
VRCL D2(82)	Clear VR	S	A6C5	IZ
VRRS GR1	Restore VR	RRE	A648	IZ XC
VRSV GR1	Save VR	RRE	A64A	IZ C
VRSVC GR1	Save Changed VR	RRE	A649	IZ PC
VS VR1,VR3,RS2(RT2)	Subtract	VST	A421	IM
VSD VR1,VR3,RS2(RT2)	Subtract (L)	VST	A411	IM
VSDQ VR1,FR3,VR2	Subtract (L)	QV	A591	IM
VSDR VR1,VR3,VR2	Subtract (L)	VV	A511	IM
VSDS VR1,FR3,RS2(RT2)	Subtract (L)	QST	A491	IM
VSE VR1,VR3,RS2(RT2)	Subtract (S)	VST	A401	IM
VSEQ VR1,FR3,VR2	Subtract (S)	QV	A581	IM
VSER VR1,VR3,VR2	Subtract (S)	VV	A501	IM
VSES VR1,FR3,RS2(RT2)	Subtract (S)	QST	A481	IM
VSLL VR1, VR3,D2(B2)	Shift Left Single Logical	RSE	E425	IM
VSPSD VR1,FR2	Sum Partial Sums (L)	VR	A61A	ipc
VSQ VR1,GR3,VR2	Subtract	QV	A5A1	IM
VSR VR1,VR3.VR2	Subtract	VV	A521	IM
VSRL VR1,VRa,D2(B2)	Shift Right Single Logical	RSE	E424	IM
VSRRS D2(B2)	Restore VSR	S	A6C2	IZ X
VS RSV D2(B2)	Save VSR	S	A6C0	NO X
VSS VR1,GR3,RS2(RT2)	Subtract	QST	A4A1	IM
VST VR1,RS2(RT2)	Store	VST	A40D	IC
VSTD VR1,RS2(RT2)	Store (L)	VST	A41D	IC
VSTE VR1,RS2(RT2)	Store (S)	VST	A40D	IC
VSTH VR1,RS2(RT2)	Store Halfword	VST	A42D	IC
VSTI VR1,VR3,D2(B2)	Store Indirect	RSE	E401	IC
VSTID VR1,VR3,D2(B2)	Store Indirect (L)	RSE	R411	IC
VSTIE VR1,VR3,D2(B2)	Store Indirect (S)	RSE	R401	IC
VSTK VR1,RS2(RT2)	Store Compressed	VST	A40F	IC
VSTKD VR1,RS2(RT2)	Store Compressed (L)	VST	A41F	IC
VSTKE VR1,RS2(RT2)	Store Compressed (S)	VST	A40F	IC
VSTM VR1,RS2(RT2)	Store Matched	VST	A40E	IC
VSTMD VR1,RS2(RT2)	Store Matched (L)	VST	A41E	IC
VSTME VR1,RS2(RT2)	Store Matched (S)	VST	A40E	IC
VSTVM RS2	Store VMR	VS	A682	NC
VSTVP D2(B2)	Store Vector Parameters	S	A6C8	NO
VSVMM D2(B2)	Set Vector Mask Mode	S	A6C6	NO
VTVM	Test VMR	RRE	A640	NC C
VX VR1, VR3,RS2(RT2)	Exclusive OR	VST	A426	IM
VXEL VR1 ,GR3,GR2	Extract Element	VR	A629	N1
VXELD VR1,FR3,GR2	Extract Element (L)	VR	A619	N1
VXELE VR1,FR3,GR2	Extract Element (S)	VR	A609	N1
VXQ VR1,GR3,VR2	Exclusive OR	QV	A5A6	IM
VXR VR1,VR3,VR2	Exclusive OR	VV	A526	IM
VXS VR1,GR3,RS2(RT2)	Exclusive OR	QST	A4A6	IM
VXVC GR1	Extract VCT	RRE	A644	NO
VXVM RS2	Exclusive OR to VMR	VS	A686	NC
VXVMM GR1	Extract Vector Mask Mode	RRE	A646	NO
VZPSO VR1	Zero Partial Sums (L)	VR	A61B	IP
WRD D1(B1),I2	Write Direct	SI	84	P
X R1,D2(X2,B2)	Exclusive OR	RX	57	C
XC D1(L,B1),D2(B2)	Exclusive OR	SS	D7	C
XI D1(B1),I2	Exclusive OR	SI	97	C
XR R1,R2	Exclusive OR	RR	17	C
ZAP D1(L1,B1),D2(L2,B2)	Zero and Add	SS	F8	C
Model-dependent	Diagnose	-	83	PY

c. Condition code set.
i. Interruptible instruction.
n. New condition code loaded.
p. Privileged instruction.
q. Semiprivileged instruction.
x. Execution in problem state and supervisor state differs.
y. Condition code may be set.

Floating-point operand lengths: Notes:
(E) Extended source and result.
(E/L) Extended source, long result.
(L/E) Long source, extended result.
(L) Long source and result.
(L/S) Long source, short result.
(S/L) Short source, long result.
(S) Short source and result.

Class (for instructions subject to vector-control bit, CR 0 bit 14)
IC: Interruptible; IVCT - VIX) elements processed.
IG: Interruptible; either (bit count in a general register) elements
or (section-size - VIX) elements processed, whichever is fewer.
IM: Interruptible; (VCT - VIX) elements processed, vector-mask mode.
IP: Interruptible; (partial-sum-number - VIX) elements processed.
IZ: Interruptible; (section-size) elements processed.
NC: Not interruptible; (VCT) elements processed.
NZ: Not interruptible; (section-size) elements processed.
NO: Not interruptible; no elements processed (VSRIVAC housekeeping).
N1: Not interruptible; one element processed.

Bases 1

Ctrl	Dec	Neg	Hex	Oct	Binary	Asc
NULL	0	0	00	###	00000000	
SOH	1	-255	01	###	00000001	☺
STX	2	-254	02	###	00000010	☻
ETX	3	-253	03	###	00000011	♥
EOT	4	-252	04	###	00000100	♦
ENQ	5	-251	05	###	00000101	♣
ACK	6	-250	06	###	00000110	♠
BEL	7	-249	07	###	00000111	•
BS	8	-248	08	###	00001000	◘
TAB	9	-247	09	###	00001001	○
LF	10	-246	0A	###	00001010	◙
VT	11	-245	0B	###	00001011	♂
FF	12	-244	0C	###	00001100	♀
CR	13	-243	0D	###	00001101	♪
SO	14	-242	0E	###	00001110	♫
SI	15	-241	0F	###	00001111	☼
DLE	16	-240	10	###	00010000	►
DC1	17	-239	11	###	00010001	◄
DC2	18	-238	12	###	00010010	↕
DC3	19	-237	13	###	00010011	‼
DC4	20	-236	14	###	00010100	¶
NAK	21	-235	15	###	00010101	§
SYN	22	-234	16	###	00010110	▬
ETB	23	-233	17	###	00010111	↨
CAN	24	-232	18	###	00011000	↑
EN	25	-231	19	###	00011001	↓
SUB	26	-230	1A	###	00011010	→
ESC	27	-229	1B	###	00011011	←
DS	28	-228	1C	###	00011100	∟
GS	29	-227	1D	###	00011101	↔
RS	30	-226	1E	###	00011110	▲
US	31	-225	1F	###	00011111	▼
	32	-224	20	040	00100000	
	33	-223	21	041	00100001	!
	34	-222	22	042	00100010	"
	35	-221	23	043	00100011	#
	36	-220	24	044	00100100	$
	37	-219	25	045	00100101	%
	38	-218	26	046	00100110	&
	39	-217	27	047	00100111	'
	40	-216	28	050	00101000	(
	41	-215	29	051	00101001	)
	42	-214	2A	052	00101010	*
	43	-213	2B	053	00101011	+
	44	-212	2C	054	00101100	,
	45	-211	2D	055	00101101	-
	46	-210	2E	056	00101110	.
	47	-209	2F	057	00101111	/
	48	-208	30	060	00110000	0
	49	-207	31	061	00110001	1
	50	-206	32	062	00110010	2
	51	-205	33	063	00110011	3
	52	-204	34	064	00110100	4
	53	-203	35	065	00110101	5
	54	-202	36	066	00110110	6
	55	-201	37	067	00110111	7
	56	-200	38	070	00111000	8
	57	-199	39	071	00111001	9
	58	-198	3A	072	00111010	:
	59	-197	3B	073	00111011	;
	60	-196	3C	074	00111100	<
	61	-195	3D	075	00111101	=
	62	-194	3E	076	00111110	>
	63	-193	3F	077	00111111	?
	64	-192	40	100	01000000	@
	65	-191	41	101	01000001	A
	66	-190	42	102	01000010	B
	67	-189	43	103	01000011	C
	68	-188	44	104	01000100	D
	69	-187	45	105	01000101	E
	70	-186	46	106	01000110	F
	71	-185	47	107	01000111	G
	72	-184	48	110	01001000	H
	73	-183	49	111	01001001	I
	74	-182	4A	112	01001010	J
	75	-181	4B	113	01001011	K
	76	-180	4C	114	01001100	L
	77	-179	4D	115	01001101	M
	78	-178	4E	116	01001110	N
	79	-177	4F	117	01001111	O
	80	-176	50	120	01010000	P
	81	-175	51	121	01010001	Q
	82	-174	52	122	01010010	R
	83	-173	53	123	01010011	S
	84	-172	54	124	01010100	T
	85	-171	55	125	01010101	U
	86	-170	56	126	01010110	V
	87	-169	57	127	01010111	W
	88	-168	58	130	01011000	X
	89	-167	59	131	01011001	Y
	90	-166	5A	132	01011010	Z
	91	-165	5B	133	01011011	[
	92	-164	5C	134	01011100	\
	93	-163	5D	135	01011101	]
	94	-162	5E	136	01011110	^
	95	-161	5F	137	01011111	_
	96	###	60	140	01100000	`
	97	###	61	141	01100001	a
	98	###	62	142	01100010	b
	99	###	63	143	01100011	c
	100	###	64	144	01100100	d
	101	###	65	145	01100101	e
	102	###	66	146	01100110	f
	103	###	67	147	01100111	g
	104	###	68	150	01101000	h
	105	###	69	151	01101001	i
	106	###	6A	152	01101010	j
	107	###	6B	153	01101011	k
	108	###	6C	154	01101100	l
	109	###	6D	155	01101101	m
	110	###	6E	156	01101110	n
	111	###	6F	157	01101111	o
	112	###	70	160	01110000	p
	113	###	71	161	01110001	q
	114	###	72	162	01110010	r
	115	###	73	163	01110011	s
	116	###	74	164	01110100	t
	117	###	75	165	01110101	u
	118	###	76	166	01110110	v
	119	###	77	167	01110111	w
	120	###	78	170	01111000	x
	121	###	79	171	01111001	y
	122	###	7A	172	01111010	z
	123	###	7B	173	01111011	{
	124	###	7C	174	01111100	|
	125	###	7D	175	01111101	}
	126	###	7E	176	01111110	~
	127	###	7F	177	01111111	⌂

Dec	Neg	Hex	Oct	Binary	Asc
128	-128	80	200	10000000	Ç
129	-127	81	201	10000001	ü
130	-126	82	202	10000010	é
131	-125	83	203	10000011	â
132	-124	84	204	10000100	ä
133	-123	85	205	10000101	à
134	-122	86	206	10000110	å
135	-121	87	207	10000111	ç
136	-120	88	210	10001000	ê
137	-119	89	211	10001001	ë
138	-118	8A	212	10001010	è
139	-117	8B	213	10001011	ï
140	-116	8C	214	10001100	î
141	-115	8D	215	10001101	ì
142	-114	8E	216	10001110	Ä
143	-113	8F	217	10001111	Å
144	-112	90	220	10010000	É
145	-111	91	221	10010001	æ
146	-110	92	222	10010010	Æ
147	-109	93	223	10010011	ô
148	-108	94	224	10010100	ö
149	-107	95	225	10010101	ò
150	-106	96	226	10010110	û
151	-105	97	227	10010111	ù
152	-104	98	230	10011000	ÿ
153	-103	99	231	10011001	Ö
154	-102	9A	232	10011010	Ü
155	-101	9B	233	10011011	¢
156	-100	9C	234	10011100	£
157	-99	9D	235	10011101	¥
158	-98	9E	236	10011110	₧
159	-97	9F	237	10011111	ƒ
160	-96	A0	240	10100000	á
161	-95	A1	241	10100001	í
162	-94	A2	242	10100010	ó
163	-93	A3	243	10100011	ú
164	-92	A4	244	10100100	ñ
165	-91	A5	245	10100101	Ñ
166	-90	A6	246	10100110	ª
167	-89	A7	247	10100111	º
168	-88	A8	250	10101000	¿
169	-87	A9	251	10101001	⌐
170	-86	AA	252	10101010	¬
171	-85	AB	253	10101011	½
172	-84	AC	254	10101100	¼
173	-83	AD	255	10101101	¡
174	-82	AE	256	10101110	«
175	-81	AF	257	10101111	»
176	-80	B0	260	10110000	░
177	-79	B1	261	10110001	▒
178	-78	B2	262	10110010	▓
179	-77	B3	263	10110011	│
180	-76	B4	264	10110100	┤
181	-75	B5	265	10110101	╡
182	-74	B6	266	10110110	╢
183	-73	B7	267	10110111	╖
184	-72	B8	270	10111000	╕
185	-71	B9	271	10111001	╣
186	-70	BA	272	10111010	║
187	-69	BB	273	10111011	╗
188	-68	BC	274	10111100	╝
189	-67	BD	275	10111101	╜
190	-66	BE	276	10111110	╛
191	-65	BF	277	10111111	┐
192	-64	C0	300	11000000	└
193	-63	C1	301	11000001	┴
194	-62	C2	302	11000010	┬
195	-61	C3	303	11000011	├
196	-60	C4	304	11000100	─
197	-59	C5	305	11000101	┼
198	-58	C6	306	11000110	╞
199	-57	C7	307	11000111	╟
200	-56	C8	310	11001000	╚
201	-55	C9	311	11001001	╔
202	-54	CA	312	11001010	╩
203	-53	CB	313	11001011	╦
204	-52	CC	314	11001100	╠
205	-51	CD	315	11001101	═
206	-50	CE	316	11001110	╬
207	-49	CF	317	11001111	╧
208	-48	D0	320	11010000	╨
209	-47	D1	321	11010001	╤
210	-46	D2	322	11010010	╥
211	-45	D3	323	11010011	╙
212	-44	D4	324	11010100	╘
213	-43	D5	325	11010101	╒
214	-42	D6	326	11010110	╓
215	-41	D7	327	11010111	╫
216	-40	D8	330	11011000	╪
217	-39	D9	331	11011001	┘
218	-38	DA	332	11011010	┌
219	-37	DB	333	11011011	█
220	-36	DC	334	11011100	▄
221	-35	DD	335	11011101	▌
222	-34	DE	336	11011110	▐
223	-33	DF	337	11011111	▀
224	-32	E0	340	11100000	α
225	-31	E1	341	11100001	ß
226	-30	E2	342	11100010	Γ
227	-29	E3	343	11100011	π
228	-28	E4	344	11100100	Σ
229	-27	E5	345	11100101	σ
230	-26	E6	346	11100110	µ
231	-25	E7	347	11100111	τ
232	-24	E8	350	11101000	Φ
233	-23	E9	351	11101001	Θ
234	-22	EA	352	11101010	Ω
235	-21	EB	353	11101011	δ
236	-20	EC	354	11101100	∞
237	-19	ED	355	11101101	φ
238	-18	EE	356	11101110	ε
239	-17	EF	357	11101111	∩
240	-16	F0	360	11110000	≡
241	-15	F1	361	11110001	±
242	-14	F2	362	11110010	≥
243	-13	F3	363	11110011	≤
244	-12	F4	364	11110100	⌠
245	-11	F5	365	11110101	⌡
246	-10	F6	366	11110110	÷
247	-9	F7	367	11110111	≈
248	-8	F8	370	11111000	°
249	-7	F9	371	11111001	∙
250	-6	FA	372	11111010	·
251	-5	FB	373	11111011	√
252	-4	FC	374	11111100	ⁿ
253	-3	FD	375	11111101	²
254	-2	FE	376	11111110	■
255	-1	FF	377	11111111	

Bases 2

Dec	Neg	Hex	Oct	Binary
0	0	0000	000000	00000000 00000000
1	-65535	0001	000001	00000000 00000001
2	-65534	0002	000002	00000000 00000010
4	-65532	0004	000004	00000000 00000100
8	-65528	0008	000010	00000000 00001000
16	-65520	0010	000020	00000000 00010000
32	-65504	0020	000040	00000000 00100000
64	-65472	0040	000100	00000000 01000000
128	-65408	0080	000200	00000000 10000000
192	-65344	00C0	000300	00000000 11000000
256	-65280	0100	000400	00000001 00000000
320	-65216	0140	000500	00000001 01000000
384	-65152	0180	000600	00000001 10000000
448	-65088	01C0	000700	00000001 11000000
512	-65024	0200	001000	00000010 00000000
576	-64960	0240	001100	00000010 01000000
640	-64896	0280	001200	00000010 10000000
704	-64832	02C0	001300	00000010 11000000
768	-64768	0300	001400	00000011 00000000
832	-64704	0340	001500	00000011 01000000
896	-64640	0380	001600	00000011 10000000
960	-64576	03C0	001700	00000011 11000000
1,024	-64512	0400	002000	00000100 00000000
1,088	-64448	0440	002100	00000100 01000000
1,152	-64384	0480	002200	00000100 10000000
1,216	-64320	04C0	002300	00000100 11000000
1,280	-64256	0500	002400	00000101 00000000
1,344	-64192	0540	002500	00000101 01000000
1,408	-64128	0580	002600	00000101 10000000
1,472	-64064	05C0	002700	00000101 11000000
1,536	-64000	0600	003000	00000110 00000000
1,600	-63936	0640	003100	00000110 01000000
1,664	-63872	0680	003200	00000110 10000000
1,728	-63808	06C0	003300	00000110 11000000
1,792	-63744	0700	003400	00000111 00000000
1,856	-63680	0740	003500	00000111 01000000
1,920	-63616	0780	003600	00000111 10000000
1,984	-63552	07C0	003700	00000111 11000000
2,048	-63488	0800	004000	00001000 00000000

Dec	Neg	Hex	Oct	Binary
0	0	0000	000000	00000000 00000000
2,048	-63488	0800	004000	00001000 00000000
4,096	-61440	1000	010000	00010000 00000000
6,144	-59392	1800	014000	00011000 00000000
8,192	-57344	2000	020000	00100000 00000000
10,240	-55296	2800	024000	00101000 00000000
12,288	-53248	3000	030000	00110000 00000000
14,336	-51200	3800	034000	00111000 00000000
16,384	-49152	4000	040000	01000000 00000000
18,432	-47104	4800	044000	01001000 00000000
20,480	-45056	5000	050000	01010000 00000000
22,528	-43008	5800	054000	01011000 00000000
24,576	-40960	6000	060000	01100000 00000000
26,624	-38912	6800	064000	01101000 00000000
28,672	-36864	7000	070000	01110000 00000000
30,720	-34816	7800	074000	01111000 00000000
32,768	-32768	8000	100000	10000000 00000000
34,816	-30720	8800	104000	10001000 00000000
36,864	-28672	9000	110000	10010000 00000000
38,912	-26624	9800	114000	10011000 00000000
40,960	-24576	A000	120000	10100000 00000000
43,008	-22528	A800	124000	10101000 00000000
45,056	-20480	B000	130000	10110000 00000000
47,104	-18432	B800	134000	10111000 00000000
49,152	-16384	C000	140000	11000000 00000000
51,200	-14336	C800	144000	11001000 00000000
53,248	-12288	D000	150000	11010000 00000000
55,296	-10240	D800	154000	11011000 00000000
57,344	-8192	E000	160000	11100000 00000000
59,392	-6144	E800	164000	11101000 00000000
61,440	-4096	F000	170000	11110000 00000000
63,488	-2048	F800	174000	11111000 00000000
65,535	-1	FFFF	177777	11111111 11111111

16,777,215	FFFFFF	77777777	24 Bit
4,294,967,295	FFFFFFFF	3.7778E+10	32 Bit

Colors & Resolutions

Screen	Bytes	Hex	Size (K)
256x192 256 color	49,152	C000	48K
256x192 16 color	24,576	6000	24K
256x192 8 color	18,432	4800	18K
256x192 4 color	12,288	3000	12K
256x192 2 color	6,144	1800	6K

Screen	Bytes	Hex	Size (K)
320x200 256 color	64,000	FA00	64K
320x200 16 color	32,000	7D00	32K
320x200 8 color	24,000	5DC0	24K
320x200 4 color	16,000	3E80	16K
320x200 2 color	8,000	1F40	8K

Screen	Bytes	Hex	Size (K)
32x32 Word Tiles	2,048	0800	2K
32x24 Word Tiles	1,536	0600	1.5K
32x32 Byte Tiles	1,024	0400	1K
32x24 Byte Tiles	768	0300	0.7K

Tile/Sprite	Bytes	Hex
8x8 2 color	8	08
8x8 4 color	16	10
8x8 8 color	24	18
8x8 16 color	32	20
8x8 256 color	64	40
16x16 2 color	32	20
16x16 4 color	64	40
16x16 8 color	96	60
16x16 16 color	128	80
16x16 256 color	256	100

Common Color combinations

Color	**-RGB**	
Red	-F00	🟥
Green	-0F0	🟩
Blue	-00F	🟦
Cyan	-FF0	🟦
Magenta	-F0F	🟪
Yellow	-FF0	🟨
Black	-000	⬛
Grey	-888	◼
White	-FFF	⬜
Orange	-F80	🟧
Pink	-F88	🟥
Lilac	-88F	🟪
Sky Blue	-08F	🟦
Purple	-80F	🟪

Systems such as the ZX Spectrum, Campters Lynx Fujitsu FM-7 and Sinclair QL use a palette based on Combinations of 3 primary color channel bitplanes:

Color	**Number**	**-GRB**	
Black	0	-000	⬛
Blue	1	-001	🟦
Red	2	-010	🟥
Magenta	3	-011	🟪
Green	4	-100	🟩
Cyan	5	-101	🟦
Yellow	6	-110	🟨
White	7	-111	⬜

Trigonometry

Deg	Radians		SIN	COS	TAN	Value	ASIN	ACOS	ATAN	CSC	SEC	COT	Byte Rad	Byte Sin
0	0	0.000	0.000	1.000	0.000	0.000	0.000	1.571	0.000	#NUM!	1.000	#NUM!	0	0
15		0.262	0.259	0.966	0.268	0.259	0.262	1.309	0.253	3.864	1.035	3.732	11	33
30	π/6	0.524	0.500	0.866	0.577	0.500	0.524	1.047	0.464	2.000	1.155	1.732	21	64
45	π/4	0.785	0.707	0.707	1.000	0.707	0.785	0.785	0.615	1.414	1.414	1.000	32	90
60	π/3	1.047	0.866	0.500	1.732	0.866	1.047	0.524	0.714	1.155	2.000	0.577	43	110
75		1.309	0.966	0.259	3.732	0.966	1.309	0.262	0.768	1.035	3.864	0.268	53	123
90	π/2	1.571	1.000	0.000	###	1.000	1.571	0.000	0.785	1.000	###	0.000	64	127
105		1.833	0.966	-0.259	-3.732	0.966	1.309	0.262	0.768	1.035	-3.864	-0.268	75	123
120	2π/3	2.094	0.866	-0.500	-1.732	0.866	1.047	0.524	0.714	1.155	-2.000	-0.577	85	110
135	3π/4	2.356	0.707	-0.707	-1.000	0.707	0.785	0.785	0.615	1.414	-1.414	-1.000	96	90
150	5π/6	2.618	0.500	-0.866	-0.577	0.500	0.524	1.047	0.464	2.000	-1.155	-1.732	107	64
165		2.880	0.259	-0.966	-0.268	0.259	0.262	1.309	0.253	3.864	-1.035	-3.732	117	33
180	π	3.142	0.000	-1.000	0.000	0.000	0.000	1.571	0.000	###	-1.000	###	128	0
195		3.403	-0.259	-0.966	0.268	-0.259	-0.262	1.833	-0.253	-3.864	-1.035	3.732	139	-33
210		3.665	-0.500	-0.866	0.577	-0.500	-0.524	2.094	-0.464	-2.000	-1.155	1.732	149	-64
225		3.927	-0.707	-0.707	1.000	-0.707	-0.785	2.356	-0.615	-1.414	-1.414	1.000	160	-90
240		4.189	-0.866	-0.500	1.732	-0.866	-1.047	2.618	-0.714	-1.155	-2.000	0.577	171	-110
255		4.451	-0.966	-0.259	3.732	-0.966	-1.309	2.880	-0.768	-1.035	-3.864	0.268	181	-123
270	3π/2	4.712	-1.000	0.000	###	-1.000	-1.571	3.142	-0.785	-1.000	###	0.000	192	-127
285		4.974	-0.966	0.259	-3.732	-0.966	-1.309	2.880	-0.768	-1.035	3.864	-0.268	203	-123
300		5.236	-0.866	0.500	-1.732	-0.866	-1.047	2.618	-0.714	-1.155	2.000	-0.577	213	-110
315		5.498	-0.707	0.707	-1.000	-0.707	-0.785	2.356	-0.615	-1.414	1.414	-1.000	224	-90
330		5.760	-0.500	0.866	-0.577	-0.500	-0.524	2.094	-0.464	-2.000	1.155	-1.732	235	-64
345		6.021	-0.259	0.966	-0.268	-0.259	-0.262	1.833	-0.253	-3.864	1.035	-3.732	245	-33
360	2π	6.283	0.000	1.000	0.000	0.000	0.000	1.571	0.000	###	1.000	###	256	0
375		6.5450	0.2588	0.9659	0.2679	0.2588	0.2618	1.3090	0.2533					
390		6.8068	0.5000	0.8660	0.5774	0.5000	0.5236	1.0472	0.4636					

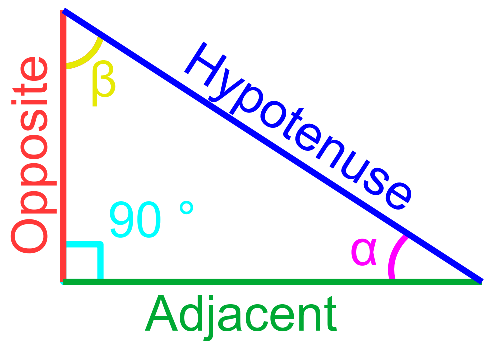
Cos (X) = Sin (X + 90°)
Sin (X) = Cos (X - 90°)

Csc (X) = 1/Sin (X)
Sec (X) = 1/Cos (X)

Cot (X) = 1/Tan (X)
Cot (X) = Cos (X) / Sin (X)

degrees = radians × 180° / π
radians = degrees × π / 180°
90 Degrees = π/2 rad

Small Angle Approximations
Sin a ~= a
Cos a ~= 1 - (a * a) / 2 ~= 1
Tan a ~= a



Trigonometry	Sin (α) = Opposite / Hypotenuse
Trigonometry	Cos (α) = Adjacent / Hypotenuse
Trigonometry	Tan (α) = Opposite / Adjacent
Trigonometry	Csc (α) = Hypotenuse / Opposite
Trigonometry	Sec (α) = Hypotenuse / Adjacent
Trigonometry	Cot (α) = Adjacent / Opposite

180 Rule	90 + α + β = 180°
Pythagoras	$H^2 = A^2 + O^2$

	Adj =	Opp =	Hyp =	α =
Trigonometry	Cos (α) * H	Sin (α) * H	O / Sin (α)	ATan (O / A)
Trigonometry	O / Tan (α)	Tan (α) * A	A / Cos (α)	ACos (A / H)
Trigonometry				ASin (O / H)
Pythagoras	$\sqrt{(H^2 - O^2)}$	$\sqrt{(H^2 - A^2)}$	$\sqrt{(A^2 + O^2)}$	
180 Rule				180 − (90 + β)

Deg	Radians		SIN	COS	TAN
0	0	0.0000	0.0000	1.0000	0.0000
15		0.2618	0.2588	0.9659	0.2679
30	π/6	0.5236	0.5000	0.8660	0.5774
45	π/4	0.7854	0.7071	0.7071	1.0000
60	π/3	1.0472	0.8660	0.5000	1.7321
75		1.3090	0.9659	0.2588	3.7321
90	π/2	1.5708	1.0000	0.0000	###
105		1.8326	0.9659	-0.2588	-3.7321
120	2π/3	2.0944	0.8660	-0.5000	-1.7321
135	3π/4	2.3562	0.7071	-0.7071	-1.0000
150	5π/6	2.6180	0.5000	-0.8660	-0.5774
165		2.8798	0.2588	-0.9659	-0.2679
180	π	3.1416	0.0000	-1.0000	-0.0000
195		3.4034	-0.2588	-0.9659	0.2679
210		3.6652	-0.5000	-0.8660	0.5774
225		3.9270	-0.7071	-0.7071	1.0000
240		4.1888	-0.8660	-0.5000	1.7321
255		4.4506	-0.9659	-0.2588	3.7321
270	3π/2	4.7124	-1.0000	-0.0000	###
285		4.9742	-0.9659	0.2588	-3.7321
300		5.2360	-0.8660	0.5000	-1.7321
315		5.4978	-0.7071	0.7071	-1.0000
330		5.7596	-0.5000	0.8660	-0.5774
345		6.0214	-0.2588	0.9659	-0.2679
360	2π	6.2832	-0.0000	1.0000	-0.0000
375		6.5450	0.2588	0.9659	0.2679
390		6.8068	0.5000	0.8660	0.5774

Value	ASIN	ACOS	ATAN	CSC	SEC	COT
0.0000	0.0000	1.5708	0.0000	#NUM!	1.0000	#NUM!
0.2588	0.2618	1.3090	0.2533	3.8637	1.0353	3.7321
0.5000	0.5236	1.0472	0.4636	2.0000	1.1547	1.7321
0.7071	0.7854	0.7854	0.6155	1.4142	1.4142	1.0000
0.8660	1.0472	0.5236	0.7137	1.1547	2.0000	0.5774
0.9659	1.3090	0.2618	0.7681	1.0353	3.8637	0.2679
1.0000	1.5708	0.0000	0.7854	1.0000	###	0.0000
0.9659	1.3090	0.2618	0.7681	1.0353	-3.8637	-0.2679
0.8660	1.0472	0.5236	0.7137	1.1547	-2.0000	-0.5774
0.7071	0.7854	0.7854	0.6155	1.4142	-1.4142	-1.0000
0.5000	0.5236	1.0472	0.4636	2.0000	-1.1547	-1.7321
0.2588	0.2618	1.3090	0.2533	3.8637	-1.0353	-3.7321
0.0000	0.0000	1.5708	0.0000	###	-1.0000	###
-0.2588	-0.2618	1.8326	-0.2533	-3.8637	-1.0353	3.7321
-0.5000	-0.5236	2.0944	-0.4636	-2.0000	-1.1547	1.7321
-0.7071	-0.7854	2.3562	-0.6155	-1.4142	-1.4142	1.0000
-0.8660	-1.0472	2.6180	-0.7137	-1.1547	-2.0000	0.5774
-0.9659	-1.3090	2.8798	-0.7681	-1.0353	-3.8637	0.2679
-1.0000	-1.5708	3.1416	-0.7854	-1.0000	###	0.0000
-0.9659	-1.3090	2.8798	-0.7681	-1.0353	3.8637	-0.2679
-0.8660	-1.0472	2.6180	-0.7137	-1.1547	2.0000	-0.5774
-0.7071	-0.7854	2.3562	-0.6155	-1.4142	1.4142	-1.0000
-0.5000	-0.5236	2.0944	-0.4636	-2.0000	1.1547	-1.7321
-0.2588	-0.2618	1.8326	-0.2533	-3.8637	1.0353	-3.7321
-0.0000	-0.0000	1.5708	-0.0000	###	1.0000	###
0.2588	0.2618	1.3090	0.2533			
0.5000	0.5236	1.0472	0.4636			

HexRad	HEXSin
0	0
11	33
21	64
32	90
43	110
53	123
64	127
75	123
85	110
96	90
107	64
117	33
128	0
139	-33
149	-64
160	-90
171	-110
181	-123
192	-127
203	-123
213	-110
224	-90
235	-64
245	-33
256	-0

MatrixA ∗ MatrixB

A Rows Must = B Cols

			Matrix B				
			A	B	C	D	E
			F	G	H	I	J
Matrix A	K	L	KA+LF	KB+LG	KC+LH	KD+LI	KE+LJ
	M	N	MA+NF	MB+NG	MC+NH	MD+NI	ME+NJ
	O	P	OA+PF	OB+PG	OC+PH	OD+PI	OE+PJ
	Q	R	QA+RF	QB+RG	QC+RH	QD+RI	QE+RJ

MatrixA + MatrixB

Must Be Same Size

A	B	C
D	E	F
G	H	I
	+	
J	K	L
M	N	O
P	Q	R
	=	
A+J	B+K	C+L
D+M	E+N	F+O
G+P	H+Q	I+R

Small Angle Approximations
Sin a ~= a
Cos a ~= 1 - (a * a) / 2 ~= 1
Tan a ~= a

Cos(X) = Sin(X+ 90degrees)
degrees = radians × 180° / π
radians = degrees × π / 180°

CSC = 1/SIN (A)
SEC = 1/COS (A)
COT = 1/TAN (A)

3D Space

	X	Y	Z	W
X	1	0	0	0
Y	0	1	0	0
Z	0	0	1	0
W	0	0	0	1

X = Across
Y = Up
Z = In
W = Transformation

Rotate around **X**

	X	Y	Z
X	1	0	0
Y	0	cos (A)	sin (A)
Z	0	-sin (A)	cos (A)

X2 = X
Y2 = Y ∗ cos (A) − Z ∗ sin (A)
Z2 = Y ∗ sin (A) + Z ∗ cos (A)

Rotate around **Y**

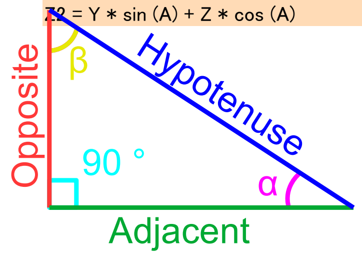
	X	Y	Z
X	cos (A)	0	−sin (A)
Y	0	1	0
Z	sin (A)	0	cos (A)

X2 = Z ∗ sin (A) + X ∗ cos (A)
Y2 = Y
Z2 = Z ∗ cos (A) − X ∗ sin (A)

Rotate around **Z**

	X	Y	Z
X	cos (A)	sin (A)	0
Y	−sin (A)	cos (A)	0
Z	0	0	1

X2 = X ∗ cos (A) − Y ∗ sin (A)
Y2 = X ∗ sin (A) + Y ∗ cos (A)
Z2 = Z



Trigonometry	Sin (α) = Opposite / Hypotenuse
Trigonometry	Cos (α) = Adjacent / Hypotenuse
Trigonometry	Tan (α) = Opposite / Adjacent
Trigonometry	Csc (α) = Hypotenuse / Opposite
Trigonometry	Sec (α) = Hypotenuse / Adjacent
Trigonometry	Cot (α) = Adjacent / Opposite

180 Rule 90 + α + β = 180°

Pythagoras $H^2 = A^2 + O^2$

Cos (X) = Sin (X + 90°)
Sin (X) = Cos (X - 90°)
Csc (X) = 1/Sin (X)
Sec (X) = 1/Cos (X)
Cot (X) = 1/Tan (X)
Cot (X) = Cos (X) / Sin (X)

90 Degrees = π/2 rad

Degrees = rad × 180 / π
Radians = deg × π / 180

	Adj =	Opp =	Hyp =	α =
Trigonometry	Cos (α) ∗ H	Sin (α) ∗ H	O / Sin (α)	ATan (O / A)
Trigonometry	O / Tan (α)	Tan (α) ∗ A	A / Cos (α)	ACos (A / H)
Trigonometry				ASin (O / H)
Pythagoras	$\sqrt{(H^2 - O^2)}$	$\sqrt{(H^2 - A^2)}$	$\sqrt{(A^2 + O^2)}$	
180 Rule				180 − (90 + β)

Rotate around **XYZ**

	X ... Roll α	Y ... Pitch β	Z ... Yaw γ
X	cos(rz) * cos(ry)	cos(rz)*sin(ry)*sin(rx) - sin(rz)*cos(rx)	cos(rz)*sin(ry)*cos(rx) + sin(rz)*sin(rx)
Y	sin(rz) * cos(ry)	sin(rz)*sin(ry)*sin(rx) + cos(rz)*cos(rx)	sin(rz)*sin(ry)*cos(rx) - cos(rz)*sin(rx)
Z	-sin(ry)	cos(ry) * sin(rx)	cos(ry) * cos(rx)

x2 = x1∗ (cosz∗cosy) + y1∗ (sinz∗cosy) − z1∗ siny
y2 = x1∗ (cosz∗siny∗sinx−sinz∗cosx) + y1∗ (sinz∗siny∗sinx+cosz∗cosx) + z1∗ (cosy∗sinx)
z2 = x1∗ (cosz∗siny∗cosx+sinz∗sinx) + y1∗ (sinz∗siny∗cosx−cosz∗sinx) + z1∗ (cosy∗cosx)

Small Angle Approx

	X	Y	Z
X	1	α	0
Y	-α	1	-β
Z	-αβ	β	1

X = X + α ∗ Y
Y = Y − α ∗ X − β ∗ Z
Z = Z + β ∗ (Y − α ∗ X)

Minsky Circle

	X	Y	Z
X	1	α	0
Y	-α	1	-β
Z	-αβ	β	1

X = X + α ∗ (Y − α ∗ X)
Y = Y − α ∗ X − β ∗ Z
Z = Z + β ∗ (Y − α∗ X − β ∗ Z)

Sources:
https://archive.org/details/Amiga3dGraphicProgrammingInBasic
https://archive.org/details/Atari_ST-3D_Graphics_Programming
https://archive.org/details/3d-math-primer-for-graphics-and-game-development-2e
https://github.com/kieranhj/elite-beebasm
https://en.wikipedia.org/wiki/Rotation_matrix
https://en.wikipedia.org/wiki/Small-angle_approximation