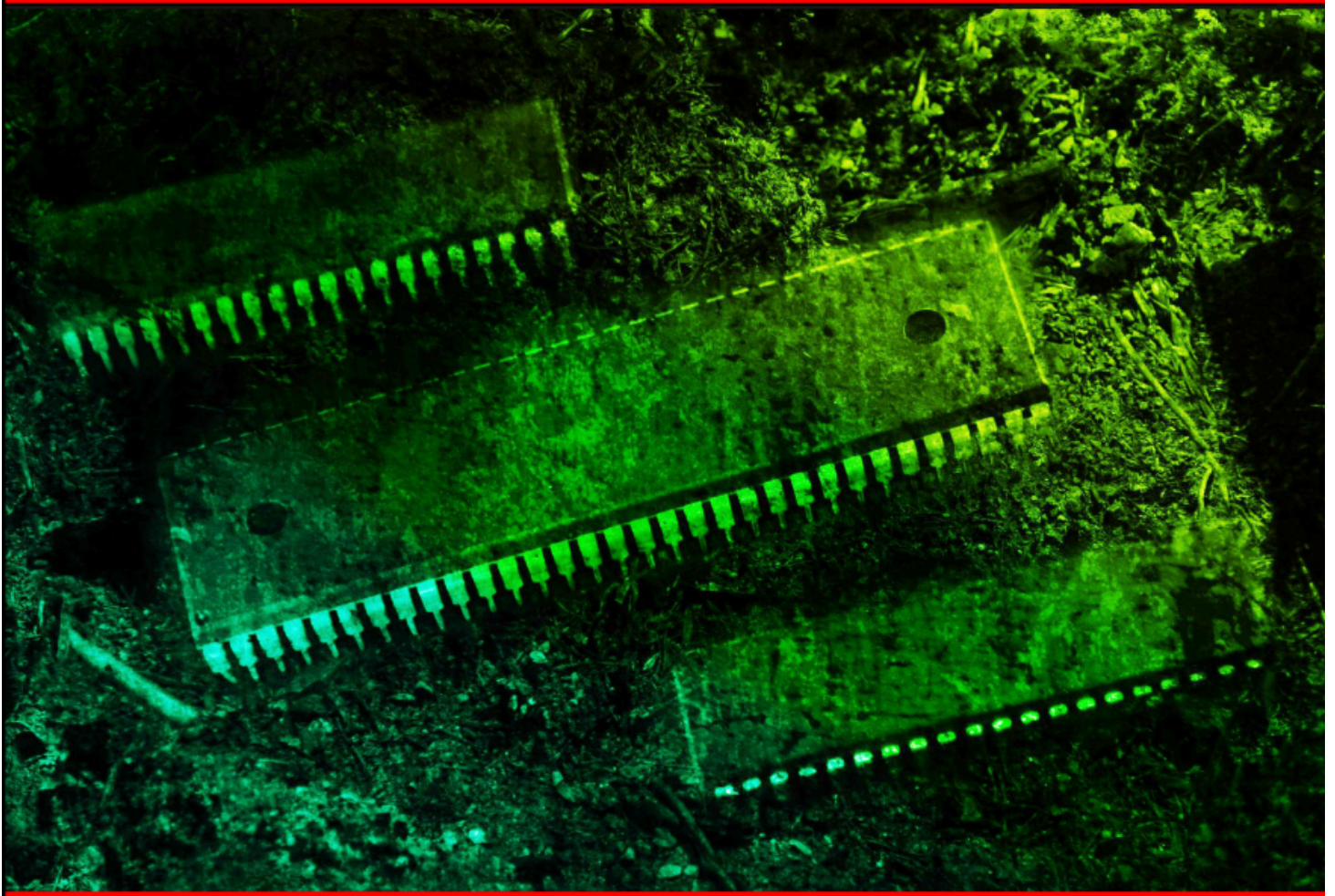


# Learn Multiplatform Assembly Programming



*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

# Learn Multiplatform Assembly Programming *with ChibiAkumas!*

Ever wanted to make your own game for an old console,  
or learn about low level programming?

Enter the world of Assembly language, and learn new things  
about classic hardware!

“Learn Multiplatform Assembly Programming... with ChibiAkumas” is an introduction to retro programming. It gives the essential technical information you'll need in a 'down to earth' style that will be more accessible to the average computer user.

Volume 1 covers the Z80, 6502, 68000, 8086 and early ARM CPUs

Volume 2 covers ARM Thumb, 65816, 6809, PDP-11 and Risc-V CPUs

Each book covers the terminology that relates to Assembly and classic hardware, an overview of the CPU and a list of the instruction set of that CPU, with clear simple descriptions.

For each CPU we'll look at some simple examples for an emulated computer or console to get you started, with details of how to compile and run them though an emulator on your Windows PC!



Get the book on Amazon now in Print or on Kindle.  
Find out more at: <http://www.chibiakumas.com/book>

## 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 rr,r1	Add 16 bit register rr1 to 16 bit register rr2.	ADD HL,BC	'r1': HL IX IY 'r2': 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	'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.	RRC 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
LL r	Shift the bits in register r Left Logically (for unsigned numbers).	LL 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

Bit	Flag	Name	Description
7	S	Sign	Positive / Negative
6	Z	Zero	Zero Flag (0=zero)
5	-		
4	H	Half Carry	Used by DAA
3	-		
2	P / V	Parity / Overflow	Detects sign change / parity in bitshift ops
1	N	Add / Subtract	Used by DAA
0	C	Carry	Carry / Borrow

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

Address	Purpose	Detail
\$0000	RST0	Reset
\$0008	RST1	
\$0010	RST2	
\$0018	RST3	
\$0020	RST4	
\$0028	RST5	
\$0030	RST6	
\$0038	RST7	Interrupt Mode 1 Interrupt Handler
\$0066	NMI	Non Maskable Interrupt





## 6502

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

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

Mnemonic	Description	Condition
<b>BCC label</b>	Branch to label	C=0
<b>BCS label</b>	Branch to label	C=1
<b>BEQ label</b>	Branch to label	Z=1
<b>BNE label</b>	Branch to label	Z=0
<b>BMI label</b>	Branch to label	N=1
<b>BPL label</b>	Branch to label	N=0
<b>BVC label</b>	Branch to label	V=0
<b>BVS label</b>	Branch to label	V=1

Bit	Flag	Name	Description
7	N	Negative	Positive / Negative
6	V	Overflow	1=True. (sign changed)
5	-		unused
4	B	Break	Interrupt was a break (in pushed flags only)
3	D	Decimal mode	1=Binary Coded Decimal mode
2	I	IRQ	1=Disable Interrupts
1	Z	Zero	Zero Flag (1=zero)
0	C	Carry	Carry (1=Carry one) / Borrow (0=Borrow one)

From	To	Purpose
\$0000	\$00FF	Zero Page
\$0100	\$01FF	Stack
...	...	...
\$FFFA	\$FFFB	NMI: Non Maskable Interrupt Vector
\$FFFC	\$FFFD	Reset Vector
\$FFFE	\$FFFF	IRQ: Interrupt Vector



	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	D	B	V	S	Absolute Long	Abs Indir Long	Direct Pg Indirect	Direct Pg Indir Long	Abs Long.X	(Long Indirect.Y)	Stack Relative	SR Indirect Indexed	
	no params	Jr	works on A	#nn	\$nn	\$nn.X	\$nn.Y	(\$0100)	(\$0100.X)	(\$0100.Y)	(\$0nnn.X)	(\$nnnn)	(\$(\$0nnn+X))	(\$(\$0nnn+Y))	0	1	2	3	4	(\$10000)	(\$1000)	(\$nn)	(\$(\$ppnn))	(\$010000.X)	(\$(\$ppnn+Y))	(\$nn.S)	(\$(\$S.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	0vE7	2	-----	-----	-----	\$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	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	-----	-----	-----	-----									
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						-----	-----	-----	-----	-----									
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												-----	-----	-----	-----	-----									
BVC	Branch if Overflow Clear		\$50 2 2-4												-----	-----	-----	-----	-----									
BVS	Branch if Overflow Set		\$70 2 2-4												-----	-----	-----	-----	-----									
CLC	Clear Carry Flag														-----	-----	-----	-----	-----									
CLD	Clear Decimal Mode		\$D8 1 2												-----	-----	-----	-----	-----									
CLI	Clear Interrupt Mask (Enable Interrupts)		\$58 1 2												-----	-----	-----	-----	-----									
CLV	Clear Overflow Flag		\$B8 1 2												-----	-----	-----	-----	-----									
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	>	-----	-----	-----	-----	\$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							>	-----	-----	-----	-----									
CPY	Compare with Index Register Y			\$C0 2 2	\$C4 2 3			\$CC 3 4							>	-----	-----	-----	-----									
DEC	Decrement (Aka DEA)		\$3A		\$C6 2 5	\$D6 2 6		\$CE 3 6	\$DE 3 7						-----	-----	-----	-----	-----									
DEX	Decrement Index Register X		\$CA 1 2												-----	-----	-----	-----	-----									
DEY	Decrement Index Register Y		\$88 1 2												-----	-----	-----	-----	-----									
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	>	-----	-----	-----	-----	\$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						-----	-----	-----	-----	-----									
INX	Increment Index Register X		\$E8 1 2												-----	-----	-----	-----	-----									
INY	Increment Index Register Y		\$C8 1 2												-----	-----	-----	-----	-----									
JMP	Jump to New Location (or JML for long)										\$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	>	-----	-----	-----	-----	\$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					-----	-----	-----	-----	-----									
LDY	Load Index Register Y			\$A0 2 2	\$A4 2 3	\$B4 2 4		\$AC 3 4	\$BC 3 4/5						-----	-----	-----	-----	-----									
LSR	Logical Shift Right		\$4A 1 2		\$46 2 5	\$56 2 6		\$4E 3 6	\$5E 3 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	>	-----	-----	-----	-----	\$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												-----	-----	-----	-----	-----									
PLP	Pull Processor Status		\$28 1 4												-----	-----	-----	-----	-----									
ROL	Rotate Left through Carry		\$2A 1 2		\$26 2 5	\$36 2 6		\$2E 3 6	\$3E 3 7						0107	2	-----	-----	-----									
ROR	Rotate Right through Carry		\$6A 1 2		\$66 2 5	\$76 2 6		\$6E 3 6	\$7E 3 7						0100	2	-----	-----	-----									
RTI	Return from Interrupt (RTI)		\$40 1 6												8	8	8	8	8									
RTS	Return from Sub (RET) (or RTL for long)		\$60 1 6												8	8	8	8	8									
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	0vE7	2	-----	-----	-----	\$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												-----	-----	-----	-----	-----									
SEI	Set Interrupt Mask (Disable Interrupts)		\$78 1 2												-----	-----	-----	-----	-----									
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												-----	-----	-----	-----	-----									
TAY	Transfer Accumulator to Index Reg Y		\$AB 1 2												-----	-----	-----	-----	-----									
TSX	Transfer Stack Pointer to X		\$BA 1 2												-----	-----	-----	-----	-----									
TXA	Transfer Index Register X to Accumulator		\$8A 1 2												-----	-----	-----	-----	-----									
TXS	Transfer X to Stack Pointer		\$9A 1 2												-----	-----	-----	-----	-----									
TYA	Transfer Index Register Y to Accumulator		\$98 1 2												-----	-----	-----	-----	-----									
BRA	Branch Relative Always (JR) (BRL for long)		\$80 2	\$82 3 4											-----	-----	-----	-----	-----									
COP	Coprocessor Enable		\$02 2 7												-----	-----	-----	-----	-----									
MVN	Block Move Next (LDIR) M.VP.M.N A bytes from MX--NY (Alters DBR)			\$54 3 7											-----	-----	-----	-----	-----									
MVP	Block Move Previous (LDDR) M.VP.M.N A bytes from MX--NY (Alters DBR)			\$44 3 7											-----	-----	-----	-----										

6502 – 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		
	no params	jr	works on A	#nn	\$nn	(\$00nn)	(\$00nn+X)	(\$00nn+Y)	\$0100	\$0100,X	\$0100,Y	(\$nnnn,X)	(\$nnnn)	(\$nn,X)	(\$nn),Y								
				8nn	(800nn)	(800nn+X)	(800nn+Y)	(80100)	(80100+X)	(80100+Y)	((800nn+X))	((8nnnn))	((800nn+X))	((800nn),Y)	ovfl 2	-	-	-	-	-	-		
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	ovfl 2	-	-	-	-	-	-	
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	-	2	-	-	-	-	-	
ASL	Arithmetic Shift Left		\$0A 1 2		\$06 2 5	\$16 2 6		\$0E 3 6	\$1E 3 7							7	2	-	-	-	-	-	
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						-	2	-	-	-	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	>	-	-	-	-	-	-	-
CPX	Compare with Index Register X				\$E0 2 2	\$E4 2 3			\$EC 3 4							>	-	-	-	-	-	-	-
CPY	Compare with Index Register Y				\$C0 2 2	\$C4 2 3			\$CC 3 4							>	-	-	-	-	-	-	-
DEC	Decrement (Aka DEA)		\$3A		\$C6 2 5	\$D6 2 6		\$CE 3 6	\$DE 3 7							-	2	-	-	-	-	-	-
DEX	Decrement Index Register X	\$CA 1 2														-	2	-	-	-	-	-	-
DEY	Decrement Index Register Y	\$88 1 2														-	2	-	-	-	-	-	-
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	-	2	-	-	-	-	-	-
INC	Increment (Aka INA)		\$1A		\$E6 2 5	\$F6 2 6		\$EE 3 6	\$FE 3 7							-	2	-	-	-	-	-	-
INX	Increment Index Register X	\$E8 1 2														-	2	-	-	-	-	-	-
INY	Increment Index Register Y	\$C8 1 2														-	2	-	-	-	-	-	-
JMP	Jump to New Location (or JML for long)							\$4C 3 3				\$7C 3	\$6C 3 5			-	-	-	-	-	-	-	-
JSR	Jump to Subroutine (or JSL for long)							\$20 3 6	\$FC 3 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	-	2	-	-	-	-	-	-
LDX	Load Index Register X				\$A2 2 2	\$A6 2 3		\$B6 2 4	\$AE 3 4							-	2	-	-	-	-	-	-
LDY	Load Index Register Y				\$A0 2 2	\$A4 2 3	\$B4 2 4		\$AC 3 4	\$BC 3 4/5						-	2	-	-	-	-	-	-
LSR	Logical Shift Right		\$4A 1 2		\$46 2 5	\$56 2 6		\$4E 3 6	\$5E 3 7							-	2	-	-	-	-	-	-
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	-	2	-	-	-	-	-	-
PHA	Push Accumulator onto Stack	\$48 1 3														-	-	-	-	-	-	-	-
PHP	Push Processor Status	\$08 1 3														-	-	-	-	-	-	-	-
PLA	Pull Accumulator from Stack	\$68 1 4														-	2	-	-	-	-	-	-
PLP	Pull Processor Status	\$28 1 4														S	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							old 7	2	-	-	-	-	-	-
ROR	Rotate Right through Carry		\$6A 1 2		\$66 2 5	\$76 2 6		\$6E 3 6	\$7E 3 7							old 0	2	-	-	-	-	-	-
RTI	Return from Interrupt (RETI)	\$40 1 6														S	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	ovfl 2	2	-	-	-	-	-	-	-
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		-	-	-	-	-	-	-	-
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														-	2	-	-	-	-	-	-
TAY	Transfer Accumulator to Index Reg Y	\$A8 1 2														-	2	-	-	-	-	-	-
TSX	Transfer Stack Pointer to X	\$BA 1 2														-	2	-	-	-	-	-	-
TXA	Transfer Index Register X to Accumulator	\$8A 1 2														-	2	-	-	-	-	-	-
TXS	Transfer X to Stack Pointer	\$9A 1 2														-	-	-	-	-	-	-	-
TYA	Transfer Index Register Y to Accumulator	\$98 1 2														-	2	-	-	-	-	-	-
BRA	Branch Relative Always (JR) (BRL for long)		\$80 2 \$82 3 4													-	-	-	-	-	-	-	-
PHX	Push X	\$DA 1														-	-	-	-	-	-	-	-
PHY	Push Y	\$5A 1														-	-	-	-	-	-	-	-
PLX	Pull X	\$FA 1														-	2	-	-	-	-	-	-
PLY	Pull Y	\$7A 1														-	2	-	-	-	-	-	-
STZ	Store Zero to address				\$64 2	\$74 2		\$9C 3	\$9E 3							-	-	-	-	-	-	-	-

>=6502  
 65C02, 65816, 6280  
 65C02, 6280 NOT 65816  
 6280  
 6280 + 65816  
 65816  
 16 bit in 65816 M=0 / X=0

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+}		-Nzvc
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
DIVU <ea>,Dn	Divide Unsigned numbers. Effectively Dn=Dn / <ea>.	DIVU #4,D1	L = L/w		-N Z V C
EOR Dn,<ea> EORI #,<ea> EORI #,CCR	Logical EOR (Exclusive OR) of bits in Dn or # with <ea>.	EOR #\$20,D1	B,W,L		-N Z V C
EORI ##,SR	is only available in Supervisor Mode.	EORI #\$F0,CCR	B		X N Z V C
EOR #,<ea>	is only available in Supervisor Mode.	EORI #\$F0,SR	B		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 'Temporary 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 <ea>,CCR	Move the contents of source <ea> to the destination <ea2>.	MOVE #15,D1	B, W, L		-N Z V C
MOVE SR,<ea> MOVE <ea>,SR	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
Sec <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			-----

<b>Bit</b>	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
<b>Flag</b>	T	-	S	-	-	I	I	I	-	-	-	X	N	Z	V	C

Flag	Name	Description
<b>T</b>	Trace bit	1 if Trace enabled (Jump to address at Trace Vector \$000024 every instruction for debugging)
<b>S</b>	Supervisor	1 if in supervisor mode, 0 in user mode
<b>I</b>	Interrupt	interrupt level (0-7)
<b>X</b>	eXtend	Used for transferring data in/out of registers in rotation
<b>N</b>	Negative	1 if top-most bit of result is 1 (meaning the value is negative)
<b>Z</b>	Zero	1 if result of previous operation was zero
<b>V</b>	Overflow	1 if arithmetic overflow occurred (last operation caused accidental sign change)
<b>C</b>	Carry	1 if the last operation resulted in a Carry or borrow

cc	Description	Flags
<b>CC</b>	carry clear	C=0
<b>CS</b>	carry set	C=1
<b>EQ</b>	Equal	Z=1
<b>GE</b>	Greater than or equal	(N=1 & V=1) or (N=0 & V=0)
<b>GT</b>	Greater than	(N=1 & V=1 & Z=0) or (N=0 & V=0 & Z=0)
<b>HI</b>	Higher than	C=0 & Z=0
<b>LE</b>	Less than or equal	Z=1 or (N=1 & V=0) or (N=0 & V=1)
<b>LS</b>	Lower than or same	C=1 or Z=1
<b>LT</b>	Less than	(N=1 and V=0) or (N=0 and V=1)
<b>MI</b>	Minus	N=1
<b>NE</b>	Not equal	Z=0
<b>PL</b>	Plus	N=0
<b>T</b>	True	=0
<b>F</b>	False	=1
<b>VC</b>	Overflow clear	V=0
<b>VS</b>	Overflow set	V=1

Cmd	Meaning	Inherent	Immediate	Direct	Extended	Indx/Indir	Relative	B	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 Condition Code		\$1C (3/2)										7
ASL	Arithmetic Shift Left			\$08 (6/2)	\$78 (7/3)	\$68 (6+2+)					*	*	*
ASLA	Arithmetic Shift Left A	\$48 (2/1)									*	*	*
ASLB	Arithmetic Shift Left B	\$58 (2/1)									*	*	*
ASR	Arithmetic Shift Right			\$07 (6/2)	\$77 (7/3)	\$67 (6+2+)					*	*	*
ASRA	Arithmetic Shift Right A	\$47 (2/1)									*	*	*
ASRB	Arithmetic Shift Right B	\$46 (2/1)									*	*	*
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+)					*	*	0
BITB	Bit Test B		\$C5 (2/2)	\$D5 (4/2)	\$F5 (5/3)	\$E5 (4+2+)					*	*	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
CLRA	Clear A										0	1	0
CLRB	Clear B	\$4F (2/1)									0	1	0
CMPA	Compare with A		\$81 (2/2)	\$91 (4/2)	\$B1 (5/3)	\$A1 (4+2+)					*	*	*
CMPB	Compare with B		\$C1 (2/2)	\$D1 (4/2)	\$F1 (5/3)	\$E1 (4+2+)					*	*	*
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+)					*	*	*
CMPLY	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)					*	*	1
COMA	Complement A	\$43 (2/1)									*	*	1
COMB	Complement B	\$53 (2/1)									*	*	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)				*	*	*
LBSC	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+)					*	*	*
NEGA	Negate A	\$40 (2/1)									*	*	*
NEGB	Negate B	\$50 (2/1)									*	*	*
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)									*	*	7
SBCA	Subtract with Carry from A		\$82 (2/2)	\$92 (4/2)	\$B2 (5/3)	\$A2 (4+2+)					*	*	*
SBCB	Subtract with Carry from B		\$C2 (2/2)	\$D2 (4/2)	\$F2 (5/3)	\$E2 (4+2+)					*	*	*
SEX	Sign Extend B into AB	\$1D (2/1)									*	*	0
STA	Store A			\$									

## 6309

Cmd	Meaning	Inherent	Immediate	Direct	Extended	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)	\$11 DB (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+)		-	-	-	-	0	*	*	*
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)		-	-	*	-	*	*	*	*
CMPE	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)					-	-	-	-	*	*	*	*
DECW	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 (3/2/5)	\$11 9D (3/2/6-27)	\$11 BD (4/27-28)	\$11 AD (3+/27+)		-	-	-	-	*	*	*	*
DIVQ	Signed Divide of Accumulator Q by 16-bit value in Memory		\$11 8E (4/3/4)	\$11 9E (3/2/5-36)	\$11 BE (4/3/6-37)	\$11 AE (3/3/6+)		-	-	-	-	*	*	*	*
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 (4/2/8)	\$11 9F (3/2/9-30)	\$11 BF (4/3/0-31)	\$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
CMPSB	Compare DS:SI to ES:DI. This command can work in bytes or words. Sets flags like CMP	REPZ CMPSB		- - - - -
CMPSW				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	Decrement Dest by one.	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 ErrorHandler		- - - - -
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		- - - - -
LODSB	Load from DS:SI into AX or AL. This command can work in bytes or words.	LODSB		- - - - -
LODSW				- - - - -
LOOP addr	Decrease CX and jump to label addr if CX is not zero.	LOOP LoopLabel		- - - - -
LOOPNZ addr	Decrease CX and jump to label addr if CX is not zero and the Zero flag is not set.	LOOPNZ LoopLabel		- - - - -
LOOPNE addr				- - - - -
LOOPZ addr	Decrease CX and jump to label addr if CX is not zero and the Zero flag is set.	LOOPZ LoopLabel		- - - - -
LOOPE addr				- - - - -
MOV dest,src	Move a value from source src to destination dest.	MOV AX,BX		- - - - -
MOVSB	Move a byte or word from DS:SI to ES:DI.	REPZ MOVSB		- - - - -
MOVSW	This command can be combined with repeat command REP, to repeat CX times.			- - - - -
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	Repeat string operation stringop while the Z flag is set and CX>0. Decrease CX each time	REPZ CMPSB		- - - - -
REPZ stringop				- - - - -
REPNE stringop	Repeat string operation stringop while the Z flag is not set and CX>0. Decrease CX each time	REPNZ CMPSB		- - - - -
REPNZ stringop				- - - - -
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 - - - - -
STOSB	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		- - - - -
XOR dest,src	Logical XOR (eXclusive OR) of bits in dest with src.	XOR AX,BX		o S Z a P C

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)	$((S \oplus O) \vee Z)=0$	
JGE / JNL	Greater or Equal / Not Less (For Signed Numbers)	$(S \oplus O)=0$	
JLE / JNG	Less than or Equal / Not Greater (For Signed Numbers)	$((S \oplus O) \vee Z)=1$	
JL / JNGE	Less / Not Greater or Equal (For Signed Numbers)	$(S \oplus O)=1$	
JNC / JAE / JNB	No Carry Above 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	





# 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 R10,R11
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 @>FEO0,R5
XOP A,n	Extend Operation	2800	9	2 2 2 2 2 2 2	Load new settings from address at vector #	XOP @>FF00,4
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 ptrs)	
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 Arithmetic	0A00	5	*****-	Shift D by B bits (0=use R0)	SLA R1,0
SRA D,B	Shift Right Arithmetic	0800	5	*****-	Shift D by B bits (0=use R0)	SRA R1,2
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				FEDCBA9876543210	FEDCBA9876543210
Instruction	Delay?	RISCV	Example		
LA dest,addr			LA	Load address	LUI \$at,>label ORI Rd,\$at,<label
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 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 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 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 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 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 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 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
LUI dest,expr	R3000		LUI	Load Upper Immediate 0xFFFF----	0 0 1 1 1 1 0 0 0 0 t t t t t t i 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 i i i
SD expr,dest				Store Doubleword	
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 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 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 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 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)
ADD rd,rs,rt	R3000	ADD	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 1 0 0 0 0 0
ADDI rt,rs,imm	R3000	ADDI	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 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 i i i
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 1 0 0 0 0 1
AND rd,rs,rt	R3000	AND	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 1 0 0 0 1 0
ANDI dest,src1,imm	R3000	ANDI	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 i i i
DIV dest,src1,src2	R3000	DIV	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
DIVU dest,src1,src2	R3000	DIVU	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 0
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 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 i i i
MUL dest,src1,src2		MUL	MUL	Multiply	MULT Rs,Rt MFLO Rd
MULO dest,src1,src2		MULH	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	MULHU	Multiply with Overflow Unsigned	MULTU Rs,Rt MFHI \$at BEQ \$at,\$0,ok ok: BREAK \$0 MFLO Rd
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 1 0 0 1 1 1
OR dest,src1,src2	R3000	OR	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 1 0 0 1 0 1
ORI dest,src1,imm	R3000	ORI	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 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
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 1 0 1 0 1 0
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 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 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:
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 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:
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 1 0 0 0 1 0
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 1 0 0 0 1 1
REM dest,src1,src2		REM	REM	Remainder (Signed)	BNE Rt,\$0,8 BREAK \$0 DIV Rs,Rt MFHI Rd
REMU dest,src1,src2		REMU	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 ORI Rd,Rd,\$a SRL \$at,Rs,32-aa SLL Rd,Rs,aa 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 ORI Rd,Rd,\$a SRL \$at,Rs,32-aa SRI Rd,Rs,aa OR Rd,Rd,\$at
SRA dest,src1,imm	R3000	SRAI	SRAI	Shift Right Arithmetic Immediate	0 0 0 0 0 0 0 0 0 0 t t t t t t d d d d d i i i i i 0 0 0 0 0 1 1
SRAV dest,src1,src2	R3000	SRA	SRA	Shift Right Arithmetic	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 1 1
SLL dest,src,imm	R3000	SLLI	SLLI	Shift Left Logical Immediate	0 0 0 0 0 0 0 0 0 0 t t t t t t d d d d d i i i i i 0 0 0 0 0 0 0
SLLV dest,src1,src2	R3000	SLL	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 1 0 0 0
SRL dest,src,imm	R3000	SRLI	SRLI	Shift Right Logical Immediate	0 0 0 0 0 0 0 0 0 0 t t t t t t d d d d d i i i i i 0 0 0 0 0 1 0
SRLV dest,src1,src2	R3000	SRL	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 1 1 0 0
ABS dest,src				Absolute value	ADDU Rd,\$0,Rs BGEZ Rs,1 SUB Rd,\$0,Rs
NEG dest,src		NEG	NEG	Negate (Signed)	SUB Rd,\$0,Rs
NEGU dest,src				Negate (Unsigned)	SUBU Rd,\$0,Rs
NOT dest,src		NOT	NOT	Not Or	NOR Rd,Rs,\$0
MOVE dest,src		MV	MV	Move	ADDU Rd,\$0,Rs
MULT src1,src2	R3000			Multiply... result in HI/LOW (leave 2 instructions before next Multi/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 1 1 0 0 0
MULTU src1,src2	R3000			Multiply (Unsigned)... result in HI/LOW (leave 2 instructions before next Multi/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 1 1 0 0 1
J addr	R3000	J	J	Jump	0 0 0 0 1 0 i
JAL addr	R3000	JAL	JAL	Jump and link	0 0 0 0 1 1 i
JALR return,reg	R3000	JALR	JALR	Jump and link Register	0 0 0 0 0 0 s s s s 0 0 0 0 0 0 d d d d d 0 0 0 0 0 1 0 0 1
JR reg	R3000	JR	JR	Jump to address in register	0 0 0 0 0 0 s s s s 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
BEQ src1, src2, label	R3000	Branch	BEQ	Branch on Equal	0 0 0 1 0 0 s s s s t t t t t t i i 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
BNE src1, src2, label	R3000	Branch	BNE	Branch on Not Equal	0 0 0 1 0 1 s s s s t t t t t t i i 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 0 0 0 0 1 i 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 0 0 0 0 0 i i 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 1 0 0 0 1 i i 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 1 0 0 0 0 i 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 0 0 0 0 0 i i 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 0 0 0 0 0 i i 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 1 0 0 0 1 i i 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 0 0 1 1 0 1
RFE				Restore from exception	0 1 0 1 0 0 0 0 0
SYSCALL		ecall		System Call	0 1 1 0 0 0
MFHI register	R3000			Move from HI to Register	0 1 0 0 0 0
MTHI register	R3000			Move to HI from Register	0 1 0 0 0 0
MFLO register	R3000			Move from LOW to register	0 1 0 0 0 0
MTLO register	R3000			Move to LOW to Register	0 1 0 0 0 1
LWCz dest,addr	R3000	LWC2 \$1,0(a0)	LWC2 \$1,0(a0)	Load Word Coprocessor Z	1 1 0 0 0 1 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 i i i
SWCz source,addr	R3000	SWC2 \$1,0(a0)	SWC2 \$1,0(a0)	Store Word Coprocessor z	1 1 1 0 0 1 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 i i i
MFCz dest-gpr,source	R3000	MFC2 a0,\$1	MFC2 a0,\$1	Move from Coprocessor z	0 1 0 0 0 0 0 0 0 0 t t t t t t d d d d d 0 0 0 0 0 0 0 0
MTCz src-gpr, destination	R3000	MTC2 a0,\$1	MTC2 a0,\$1	Move to Coprocessor z	0 1 0 0 0 0 0 0 1 0 t t t t t t d d d d d 0 0 0 0 0 0 0 0
CFCz dest-gpr, source	R3000	CFC2 a0,\$1	CFC2 a0,\$1	control from Coprocessor z	
COPz cofun	R3000	COP2 1	COP2 1	Perform Coprocessor operation cofun on coprocessor z	
CTCz src-gpr,destination	R3000	CTC2 a0,\$1	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	



Directive	<b>.comm</b> sym_name,sz,aln		emit common object to .bss section	
Directive	<b>.common</b> sym_name,sz,aln		emit common object to .bss section	
Directive	<b>.section</b> sect		emit section (if not present, default .text	[[.text,.data,.rodata,.bss]]
Directive	<b>.option</b> opt		RISC-V options	{rvc,norvc,pic,nopic,push,pop}
Directive	<b>.macro</b> name arg1 [, argn]		begin macro definition %argname to substitute	
Directive	<b>.endm</b>		end macro definition	
Directive	<b>.file</b> "filename"		emit filename FILE LOCAL symbol table	
Directive	<b>.ident</b> "string"		accepted for source compatibility	
Directive	<b>.size</b> symbol, symbol		accepted for source compatibility	
Directive	<b>.type</b> symbol, @function		accepted for source compatibility	
Pseudo	<b>NOP</b>		No operation	addi zero,zero,0
Pseudo	<b>LI</b> rd, imm	LI	Load immediate	(several expansions) (LUA+ADDI)
Pseudo	<b>LA</b> rd, symbol	LA	Load address	(several expansions)
Pseudo	<b>MV</b> rd, rs1	MOVE	Copy register	addi rd, rs, 0
Pseudo	<b>NOT</b> rd, rs1	NOT	One's complement	xori rd, rs, -1
Pseudo	<b>NEG</b> rd, rs1	NEG	Two's complement	sub rd, x0, rs
Pseudo	<b>NEGW</b> rd, rs1		Two's complement Word	subw rd, x0, rs
Pseudo	<b>SEXT.W</b> rd, rs1		Sign extend Word	addiw rd, rs, 0
Pseudo	<b>SEQZ</b> rd, rs1		Set if = zero	sltiu rd, rs, 1
Pseudo	<b>SNEZ</b> rd, rs1		Set if ≠ zero	sltu rd, x0, rs
Pseudo	<b>SLTZ</b> rd, rs1		Set if < zero	slt rd, rs, x0
Pseudo	<b>SGTZ</b> rd, rs1		Set if > zero	slt rd, x0, rs
Pseudo	<b>FMV.S</b> frd, frs1		Single-precision move	fsgnj.s frd, frs, frs
Pseudo	<b>FABS.S</b> frd, frs1		Single-precision absolute value	fsgnjx.s frd, frs, frs
Pseudo	<b>FNEG.S</b> frd, frs1		Single-precision negate	fsgnjn.s frd, frs, frs
Pseudo	<b>FMV.D</b> frd, frs1		Double-precision move	fsgnj.d frd, frs, frs
Pseudo	<b>FABS.D</b> frd, frs1		Double-precision absolute value	fsgnjx.d frd, frs, frs
Pseudo	<b>FNEG.D</b> frd, frs1		Double-precision negate	fsgjnd.d frd, frs, frs
Pseudo	<b>BEQZ</b> rs1, offset		Branch if = zero	beq rs, x0, offset
Pseudo	<b>BNEZ</b> rs1, offset		Branch if ≠ zero	bne rs, x0, offset
Pseudo	<b>BLEZ</b> rs1, offset		Branch if ≤ zero	bge x0, rs, offset
Pseudo	<b>BGEZ</b> rs1, offset		Branch if ≥ zero	bge rs, x0, offset
Pseudo	<b>BLTZ</b> rs1, offset		Branch if < zero	blt rs, x0, offset
Pseudo	<b>BGTZ</b> rs1, offset		Branch if > zero	blt x0, rs, offset
Pseudo	<b>BGT</b> rs, rt, offset		Branch if >	blt rt, rs, offset
Pseudo	<b>BLE</b> rs, rt, offset		Branch if ≤	bge rt, rs, offset
Pseudo	<b>BGTU</b> rs, rt, offset		Branch if >, unsigned	bltu rt, rs, offset
Pseudo	<b>BLEU</b> rs, rt, offset		Branch if ≤, unsigned	bltu rt, rs, offset
Pseudo	<b>J</b> offset	J	Jump	jal x0, offset
Pseudo	<b>JR</b> offset	JR	Jump register	jal x1, offset
Pseudo	<b>RET</b>		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
<b>ADCccS Rn, Rm, Op2</b>	Add With Carry.	ADC R0,R0,#4
<b>ADDccS Rn, Rm, Op2</b>	Add Op2 to Rm and store the result in Rn.	ADD R0,R0,#4
<b>ANDccS Rn, Rm, Op2</b>	Logically AND Op2 with Rm and store the result in Rn.	AND R0,R0,#4
<b>Bcc Label</b>	Branch to a relative Label.	BEQ ConditionalJump
<b>BICccS Rn, Rm, Op2</b>	Logically Bit Clear Op2 with Rm and store the result in Rn.	BIC R0,R0,#4
<b>BLcc Label</b>	Branch and Link to a relative subroutine Label.	BL TestSub
<b>CMNcc Rn, Op2</b>	Compare Negative Rn to Op2. set the flags like "ADDS Rn,Op2"	CMN R0,#4
<b>CMPcc Rn, Op2</b>	Compare Rn to Op2. set the flags, the same as "SUBS Rn,Op2"	CMP R0,#4
<b>EORccS Rn, Rm, Op2</b>	Logically Exclusive OR Op2 with Rm and store the result in Rn.	EOR R0,R0,#4
<b>LDMccadm Rn!, {Regs}</b>	Transfer range of registers {Regs} to address in Rn. Like POP	LDMFD sp!,{r0,r1,r2}
<b>LDRcc Rn, Flex</b>		
<b>LDRccB Rn, Flex</b>	Load register Rn from address Flex	LDR R0,NearLabel
<b>LDRccH Rn, Off</b>		
<b>LDRccSH Rn, Off</b>		
<b>LDRccSB Rn, Off</b>	HalfWord (16 bit), Signed Word (16 Bit) and Signed Byte (8 Bit) load	LDRSB R0,[R1,#-255]
<b>MLAccS Rn, Rm, Ro, Rp</b>	32 bit Multiplication and Add. $Rn=(Rm*Ro)+Rp$	MLA R0,R1,R2,R3
<b>MOVccS Rn, Op2</b>	Move value in Op2 into Rn.	MOV R0,#0xFF
<b>MRScc Rn,sr</b>	Move sr (either CPSR or SPSR) to register Rn.	MRS R0,SPSR
<b>MSRcc sr_f,#</b>		
<b>MSRcc sr_f,Rn</b>	Move immediate # or register into flags f of sr (either CPSR or SPSR).	MSR CPSR_F,#0
<b>MULccS Rn, Rm, Ro</b>	32 bit Multiplication. $Rn=Rm*Ro$ .	MUL R0,R1,R2
<b>MVNccS Rn, Op2</b>	Move Not. Flip all the bits of Op2 and move result into Rn.	MVN R0,#0xFF
<b>ORRccS Rn, Rm, Op2</b>	Logically OR Op2 with Rm and store the result in Rn.	ORR R0,R0,#4
<b>RSBccS Rn, Rm, Op2</b>	Reverse Subtract. This performs the calculation $Rn=Op2-Rm$ .	RSB R0,R0,#6
<b>RSCccS Rn, Rm, Op2</b>	Reverse Subtract with Carry. $Rn=(Op2-Rm)-C$ .	RSC R0,R0,#6
<b>SBCccS Rn, Rm, Op2</b>	Reverse Subtract with Carry. $Rn=(Op2-Rm)-C$ .	SBC R0,R0,#6
<b>STMccadm Rn!, {Regs}</b>	Transfer range of registers {Regs} to the address in Rn. Like PUSH	STMFD sp!,{r0,r1,r2}
<b>STRcc Rn, Flex</b>		
<b>STRccB Rn, Flex</b>	Store register Rn to address Flex.	STR r0,[r1,r2,asl #2]
<b>STRccH Rn, Off</b>		
<b>STRccSH Rn, Off</b>		
<b>STRccSB Rn, Off</b>	Half Word (16 bit), Signed half Word (16 Bit) and Signed Byte (8 Bit) store	STRSB R0,[R1,#-255]
<b>SUBccS Rn, Rm, Op2</b>	Subtract. This performs the calculation $Rn=Rm-Op2$ .	SUB R0,R0,#6
<b>SWIcc #</b>	Software Interrupt.	SWI 3
<b>SWPccB Rn, Rm, [Ro]</b>	Swap a register and memory. $Rn=[Ro]$ , $[Ro]=Rm$ .	SWPB R0,R1,[R2]
<b>TEQcc Rn, Rm, Op2</b>	Test for bitwise Equality. Set the flags like "EOR Rn,Rm,Op2"	TEQ R0,R0,#6
<b>TSTcc Rn, Rm, Op2</b>	Test bits. Set the flags like "AND Rn,Rm,Op2"	TST R0,R0,#6

ARM Addressing Modes				
Param	Mode	Format	Details	Example
Op2	Immediate	#n	Fixed value of n Can be any value made by an 8 bit immediate shifted by an even number of bits, eg 0xFF or 0xFF000000 are OK.	ADD R0,R0,#1
Op2	Register	Rn	value in register Rn	ADD R0,R0,R1
Op2	Register Shifted by Immediate	Rn, shft #n	Shift Register Rn by #n using shifter shft Options: LSL #n, LSR #n, ASR #n, ROR #n, RRX note: RRX can only shift 1 bit	MOV R0,R1,ROR #2
Op2	Register Shifted by Register	Rn, shft Rm	Shift Register by Rm using shifter shft Options: LSL Rm, LSR Rm, ASR Rm, ROR Rm	MOV R0,R1,ROR R2
Flex	Immediate offset Immediate pre-indexed	[Rn,#n] [Rn,#n]!	value from address in register Rn+n ! means Preindexed, set Rn=Rn+n	LDR R0,[R1],#n=0 LDR R0,[R1,#4] LDR R0,[R1,#-4]!
Flex	Register offset Register pre-indexed	[Rn,{-}Rm] [Rn,{-}Rm]!	value from address in register Rn+Rm ! means Preindexed, set Rn=Rn+Rm	LDR R0,[R1,-R2] LDR R0,[R1,R2]!
Flex	Scaled register offset Scaled register pre-indexed	[Rn, Rm,shft #n] [Rn, Rm,shft #n]!	value from address in register, if LSL then Rn+(Rm*#n) ! means Preindexed, set Rn=Rn+n	LDR R0,[R1,R2, LSL #2] LDR R0,[R1,R2, LSL #2]!
Flex	Immediate post-indexed	[Rn],#n	value from address in register Rn... set Rn=Rn+n (No need for ! - as it's the only purpose of the command!)	LDR R0,[R1],#4
Flex	Register post-indexed	[Rn], {-}Rm	value from address in register Rn... set Rn=Rn+Rm (No need for ! - as it's the only purpose of the command!)	LDR R0,[R1],R2 LDR R0,[R1],-R2
Flex	Scaled register post-indexed	[Rn], {-}Rm, shft #n	Shift Register by #n using shifter shft Options: LSL #n, LSR #n, ASR #n, ROR #n, RRX	LDR R0,[R1],R2,LSL #2 LDR R0,[R1],-R2,RRX

ARM Conditions		
cc code	Meaning	Flag
<b>EQ</b>	EQual	Z=1
<b>NE</b>	Not Equal	Z=0
<b>CS</b>	Carry Set	
<b>HS</b>	Higher or Same (Unsigned)	C=1
<b>CC LO</b>	Carry Clear LOver (Unsigned)	C=0
<b>MI</b>	MInus (Negative)	N=1
<b>PL</b>	PLus (Positive)	N=0
<b>VS</b>	oVerflow Set	V=1
<b>VC</b>	oVerflow Clear	V=0
<b>HI</b>	Higher (Unsigned)	C=1 and Z=0
<b>LS</b>	Lower or Same (Unsigned)	C=0 and Z=1
<b>GE</b>	Greater or Equal (Signed)	N=V
<b>LT</b>	Less Than (Signed)	N<>V
<b>GT</b>	Greater Than (Signed)	Z=0 and N=V
<b>LE</b>	Less than or Equal (Signed)	Z=1 or N<>V
<b>AL</b>	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 after	LDMIA R0!,{r1-r3,r5}				----	R0-R7
STMIA Rn!,{reglist}	Store Multiple and increment after	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 Bit)	STR r3,[sp,#4]				----	R0-R7,#= 0 to 124 (Multiples of 4)



# ARM Complete

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

## Arm 5+

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

Power PC			
Opcode	Integer Arithmetic Instructions	Details .=updateCR o=overflow c=carry e=extended m=minus one z=zero as param (R0)	R0=0?
<b>addi</b> rD,rA,SIMM	Add Immediate	The sum (rAID) + SIMM is placed into register rD.	Y
<b>addis</b> rD,rA,SIMM	Add immediate shifted	The sum (rA  0) + (SIMM    x'0000') is placed into rD.	Y
<b>add</b> rD,rA,rB	Add	The sum (rA) + (rB) is placed into register rD. <b>add</b> . rD,rA,rB <b>addo</b> rD,rA,rB <b>addo</b> . rD,rA,rB	-
<b>subf</b> rD,rA,rB	Subtract from	The sum --. (rA) + (rB) + 1 is placed into rD <b>subf</b> . rD,rA,rB <b>subfo</b> rD,rA,rB <b>subfo</b> . rD,rA,rB	-
<b>addic</b> rD,rA,SIMM	Add Immediate Carrying	The sum (rA) + SIMM is placed into register rD. <b>addic</b> . rD,rA,SIMM	-
<b>subfic</b> rD,rA,SIMM	Subtract from Immediate Carrying	The sum --. (rA) + SIMM + 1 is placed into register rD.	-
<b>addc</b> rD,rA,rB	Add Carrying	The sum (rA) + (rB) is placed into register rD. <b>addc</b> . rD,rA,rB <b>addco</b> rD,rA,rB <b>addco</b> . rD,rA,rB	-
<b>subfc</b> rD,rA,rB	Subtract from Carrying	The sum -, (rA) + (rB) + 1 is placed into register rD. <b>subfc</b> . rD,rA,rB <b>subfco</b> rD,rA,rB <b>subfco</b> . rD,rA,rB	-
<b>adde</b> rD,rA,rB	Add Extended	The sum (rA) + (rB) + XER(CA) is placed into register rD. <b>adde</b> . rD,rA,rB <b>addeo</b> rD,rA,rB <b>addeo</b> . rD,rA,rB	-
<b>subfe</b> rD,rA,rB	Subtract from Extended	The sum -, (rA) + (rB) + XER(CA) is placed into register rD. <b>subfe</b> . rD,rA,rB <b>subfeo</b> rD,rA,rB <b>subfeo</b> . rD,rA,rB	-
<b>addme</b> rD,rA	Add to Minus One Extended	The sum (rA) + XER(CA) + x'FFFFFFF' is placed into register rD. <b>addme</b> . rD,rA <b>addmeo</b> rD,rA <b>addmeo</b> . rD,rA	-
<b>subfme</b> rD,rA	Subtract from Minus One Extended	The sum ..., (rA) + XER(CA) + x'FFFFFFF' is placed into register rD. <b>subfme</b> . rD,rA <b>subfmeo</b> rD,rA <b>subfmeo</b> . rD,rA	-
<b>addze</b> rD,rA	Add to Zero Extended	The sum (rA) + XER(CA) is placed into register rD <b>addze</b> . rD,rA <b>addzeo</b> rD,rA <b>addzeo</b> . rD,rA	-
<b>subfze</b> rD,rA	Subtract from Zero Extended	The sum ..., (rA) + XER(CA) is placed into register rD. <b>subfze</b> . rD,rA <b>subfzeo</b> rD,rA <b>subfzeo</b> . rD,rA	-
<b>neg</b> rD,rA	Negate	NEGate register rA, and store the results in register rD. This converts a positive to a negative and vice versa. <b>neg</b> . rD,rA <b>nego</b> rD,rA <b>nego</b> . rD,rA	-
Opcode	Integer Compare Instructions	Details .=updateCCR o=Overflow	
<b>mulli</b> rD,rA,SIMM	Multiply Low Immediate	The low-order 32 bits of the 48-bit product (rA)*SIMM are placed into rD. Use with mulhw for full 64 bit	-
<b>mulhw</b> rD,rA,rB	Multiply High Word	The low-order 32 bits of the 64-bit product (rA) *(rB) are placed into rD. Use with mulhwu for full 64 bit <b>mulhw</b> . rD,rA,rB <b>mulwo</b> rD,rA,rB <b>mulwo</b> . rD,rA,rB	-
<b>mulhwu</b> rD,rA,rB	Multiply High Word Unsigned	The contents of rA and rB are interpreted as 32-bit signed integers. This gives the top 32 bit of the 64-bit product. <b>mulhw</b> . rD,rA,rB	-
<b>divw</b> rD,rA,rB	Divide Word	rD= rA (signed) / rB (signed). To Get Signed Remainder: divw rD,rA,rB rD = quotient mul rD,rB rD = quotient*divisor subf rD,rD,rA rD = remainder <b>divw</b> . rD,rA,rB <b>divwo</b> rD,rA,rB <b>divwo</b> . rD,rA,rB	-
<b>divwu</b> rD,rA,rB	Divide Word Unsigned	rD= rA (unsigned) / rB (unsigned). To Get Unsigned Remainder: divwu rD,rA,rB rD = quotient mul rD,rB rD = quotient*divisor subf rD,rD,rA rD = remainder <b>divwu</b> . rD,rA,rB <b>divwuo</b> rD,rA,rB <b>divwuo</b> . rD,rA,rB	-
Opcode	Integer Compare Instructions	Details i=immediate w=word	
<b>cmp</b> crfD,L,rA,rB	Compare (Signed)	CoMPare signed rA with signed rB, storing results in CR field crfD. L=Length (32/64 bits) <b>cmpi</b> crfD,L,rA,SIMM <b>cmpw</b> crfD,rA,rB (cmp crfD,O,rA,rB) <b>cmpwi</b> crfD,rA,SIMM (cmpi crfD,O,rA,SIMM)	-
<b>cmpl</b> crfD,L,rA,rB	Compare Logical (unsigned)	CoMPare unsigned rA with unsigned rB, storing results in CR field crfD. L=Length (32/64 bits) <b>cmpli</b> crfD,L,rA,UIMM <b>cmplwi</b> crfD,rA,UIMM (cmpli crfD,O,rA,UIMM) <b>cmplw</b> crfD,rA,rB (cmpl crfD,O,rA,rB)	-
Opcode	Integer Logical Instructions	Details .=updateCR i=immediate s=shifted c=complement / b=byte h=half	
<b>and</b> rA,rS,rB	AND	The contents of rS is ANDed with the contents of register rB and the result is placed into rA. <b>and</b> . rA,rS,rB <b>andi</b> . rA,rS,UIMM <b>andis</b> . rA,rS,UIMM <b>andc</b> rA,rS,rB <b>andc</b> . rA,rS,rB	-
<b>or</b> rA,rS,rB	OR	The contents of rS is ORed with the contents of rB and the result is placed into rA. <b>or</b> . rA,rS,rB <b>ori</b> rA,rS,UIMM <b>oris</b> rA,rS,UIMM <b>orc</b> rA,rS,rB <b>orc</b> . rA,rS,rB	-
<b>xor</b> rA,rS,rB	XOR	The contents of rS is XORed with the contents of rB and the result is placed into register rA. <b>xor</b> . rA,rS,rB <b>xori</b> rA,rS,UIMM <b>xoris</b> rA,rS,UIM	-
<b>nand</b> 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. <b>nand</b> . rA,rS,rB	-
<b>nor</b> rA,rS,rB	NOR	The contents of rS is ORed with the contents of rB and the one's complement of the result is placed into register rA. <b>nor</b> . rA,rS,rB	-
<b>eqv</b> rA,rS,rB	Equivalent	The contents of rS is XORed with the contents of rB and the complemented result is placed into register rA. <b>eqv</b> . rA,rS,rB	-
<b>extsb</b> rA,rS <b>extsh</b> rA,rS	Extend Sign Byte / Halfword	Register rS[24-31] are placed into rA[24-31]. Bit 24 of rS is placed into rA[0-23]. <b>extsb</b> . rA,rS <b>extsh</b> . rA,rS	-
<b>cntlzw</b> rA,rS	Count Leading Zeros Word	Count of leading zero bits of rS is placed into rA. This number ranges from 0 to 32, inclusive. <b>cntlzw</b> . rA,rS	-
Opcode	Integer Rotate Instructions	Details .=updateCR	
<b>rlwinm</b> rA,rS,SH,MB,ME	Rotate Left Word Immediate then AND with Mask	Rotate rS left by SH bits, and keep (AND) bits MB-ME, storing results in rA EG: rlwinm r3,r4,8,24,31 will shift R4<<8 and keep bits 24-31 (0=MSB 31=LSB) <b>rlwinm</b> . rA,rS,SH,MB,ME	-
<b>rlwnm</b> rA,rS,rB,MB,ME	Rotate Left Word then AND with Mask	Rotate rS left by rB bits, and keep (AND) bits MB-ME, storing results in rA EG: rlwnm r3,r4,8,24,31 will shift R4<<8 and keep bits 24-31 (0=MSB 31=LSB) <b>rlwnm</b> . rA,rS,rB,MB,ME	-
<b>rlwimi</b> rA,rS,SH,MB,ME	Rotate Left Word Immediate then Mask Insert	Rotate rS left by SH bits, and transfer bits MB-ME into rA (leaving other bits unchanged) EG: rlwimi r3,r4,8,24,31 will shift R4<<8 and transfer bits 24-31 (0=MSB 31=LSB) to R3, bits 0-23 of R3 are unchanged <b>rlwimi</b> . rA,rS,SH,MB,ME	-
Opcode	Integer Shift Instructions	Details .=updateCR	
<b>slw</b> rA,rS,rB	Shift Left Word	rS is shifted left rB bits. Bits shifted out are lost. new bits on the right are 0. <b>slw</b> . rA,rS,rB	-
<b>srw</b> rA,rS,rB	Shift Right Word (Unigned)	rS is shifted right rB bits. New bits on the left are 0. <b>srw</b> . rA,rS,rB	-
<b>srawi</b> rA,rS,SH	Shift Right Algebraic Word Immediate (Signed)	rS is right SH bits. Bits shifted out are lost. New bits on the left retain the old top bit, keeping the sign. XER[CA] is set if rS contains a negative number and any 1-bits are shifted out of position 3, otherwise XER(CA) is cleared. <b>srawi</b> . rA,rS,SH	-
<b>sraw</b> rA,rS,rB	Shift Right Algebraic Word (Signed)	rS is shifted right rB[26-31] bits. New bits on the left retain the old top bit, keeping the sign. 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 0. A If rB[26]=1, then rA is filled with 32 sign bits (bit 0) from rS. Condition register field CRO is set based on the value written into rA. <b>sraw</b> . rA,rS,rB	-
Opcode	Integer Load Instructions	Details b=byte h=half w=word / z=zero extend (unsigned) a=algebraic (Sign extended) u=update x=indexed	
<b>lbz</b> rD,d(rA)	Load Byte and Zero	Load and zero extend an 8 bit byte from the effective address. <b>lbzx</b> rD,rA,rB <b>lbzu</b> rD,d(rA) <b>lbzux</b> rD,rA,rB	Y
<b>lhz</b> rD,d(rA)	Load Half Word and Zero	Load and zero extend a 16 bit half word from the effective address. <b>lhzx</b> rD,rA,rB <b>lhzu</b> rD,d(rA) <b>lhzux</b> rD,rA,rB	Y
<b>lha</b> rD,d(rA)	Load Half Word Algebraic	Load and sign extend an 16 bit half word from the effective address. <b>lhax</b> rD,rA,rB <b>lhau</b> rD,d(rA) <b>lhaux</b> rD,rA,rB	Y
<b>lww</b> rD,d(rA)	Load Word and Zero	The effective address is the sum (rAIO)+d. The word in memory addressed by the EA is loaded into register rD[0-31]. <b>lwwx</b> rD,rA,rB <b>lwwu</b> rD,d(rA) <b>lwwux</b> rD,rA,rB	Y
Opcode	Integer Store Instructions	Details b=byte h=half w=word / x=indexed u=update	
<b>stb</b> rS,d(rA)	Store Byte	The effective address is the sum (rAIO) + d. Register rS[24-31] is stored into the byte in memory addressed by the EA. <b>stbx</b> rS,rA,rB <b>stbu</b> rS,d(rA) <b>stbux</b> rS,rA,rB	Y
<b>sth</b> 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. <b>sthx</b> rS,rA,rB <b>sthu</b> rS,d(rA) <b>sthux</b> rS,rA,rB	Y
<b>stw</b> 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. <b>stwx</b> rS,rA,rB <b>stwu</b> rS,d(rA) <b>stwux</b> rS,rA,rB	Y
PseudoOps	Miscellaneous Simplified Mnemonics	Details	
<b>nop</b>	No-Op	(equivalent to ori 0,0,0)	-
<b>li</b> 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	Load Effective address of Base+Offset (equivalent to addi rD,rA,SIMM)	-
mr rA,rS	Move Register	(equivalent to or rA,rS,rS)	-
not rA,rS	Complement Register	(equivalent to nor rA,rS,rS)	-
sub rD,rA,rB		Equivalent to subf rD,rB,rA	-
subi rD,rA,SIMM		Equivalent to addi rD,rA,-SIMM	-
<b>Opcode</b>	<b>Integer Load and Store with Byte Reversal Instructions</b>	<b>Details</b>	
lhrx rD,rA,rB	Load Half Word Byte-Reverse Indexed	Load unsigned Endian-Reversed 16 bit half into from (rA+rB). Byte reversed version of LHZX rD,rA,rB	Y
lwx rD,rA,rB	Load Word Byte-Reverse Indexed	Load Endian-Reversed word into rD from (rA+rB). This is the Byte reversed version of LWZX rD,rA,rB	Y
sthrx rS,rA,rB	Store Half Word Byte-Reverse Indexed	Store a Endian-Reversed 16 bit half from rD into (rA+rB). Byte reversed version of STHX rD,rA,rB	Y
stwx rS,rA,rB	Store Word Byte-Reverse Indexed	Store a Endian-Reversed word from rD into (rA+rB). This is the Byte reversed version of STWX rD,rA,rB	Y
<b>Opcode</b>	<b>Integer Load and Store Multiple Instructions</b>	<b>Details</b>	
lmw rD,d(rA)	Load Multiple Word (POP)	The effective address is the sum (rAIO)+d. n= 32-rD. n consecutive words starting at EA are loaded into GPRs rD through 31. EA should be 32 bit aligned. Used as stack POP	Y
stmw rS,d(rA)	Store Multiple Word (PUSH)	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. EA should be 32 bit aligned. Used as stack PUSH	Y
<b>Opcode</b>	<b>Integer Move String Instructions</b>	<b>Details</b>	
lswi rD,rA,NB	Load String Word Immediate	Load NB bytes from source address rA to registers rD+EG: lswi r4,r3,7#; Load 7 bytes into R4+ from (R3)	Y
lswx rD,rA,rB	Load String Word Indexed	Load XER:ByteCount (low 6 bits) bytes into rD+ from (rA+rB)EG: lswx r5,r3,r4#; Load XER:ByteCount (low 6 bits) bytes into R5+ from (R3+R4)	Y
stswi rS,rA,NB	Store String Word Immediate	Store NB bytes to destination address rA from registers rD+EG: stswi r4,r3,7#; Store 7 bytes from R4+ to (R3)	Y
stswx rS,rA,rB	Store String Word Indexed	Load XER:ByteCount (low 6 bits) bytes into rD+ from (rA+rB)EG: stswx r5,r3,r4#; Store XER:ByteCount (low 6 bits) bytes from R5+ to (R3+R4)	Y
<b>Opcode</b>	<b>Branch Instructions</b>	<b>Details</b>	
b imm_addr	Branch.	Branch to the address imm_addr	-
ba imm_addr	Branch Absolute.	Branch to the absolute address specified.	-
bl imm_addr	Branch then Link.	Branch to subroutine, and put return address in the Link Register (LR).	-
bla imm_addr	Branch Absolute then Link.	Branch to subroutine at absolute address, and put return address in 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	-
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 'beifl' 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.	-
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.	-
blr	Branch to Link Register	Return from sub	-
<b>Opcode (+/--branch prediction)</b>	<b>Bcc PsuedoOp</b>	<b>Details a=Absolute l=to LinkRegister ctr=to CTR L=Link</b>	
blt target	Branch if less than	blt label blla addr bltlr bltctr bltl label bllla addr bltlr bltctrl (Psuedo for bc 12.0.Target)	-
ble target	Branch if less than or equal	ble label blea addr blelr blectr blel label blela addr blelr blectrl	-
beq target	Branch if equal	beq label beqa addr beqlr beqctr beql label beqla addr beqlr beqctrl	-
bge target	Branch if greater than or equal	bge label bgea addr bgelr bgectr bgel label bgele addr bgele bgectrl	-
bgt target	Branch if greater than	bgt label bgta addr bgtlr bgtctr bgtl label bgtle addr bgtlr bgtctrl	-
bnl target	Branch if not less than	bnl label bnla addr bnllr bnllctr bnll label bnlla addr bnllr bnllctrl	-
bne target	Branch if not equal	bne label bnea addr bnelr bnectr bnel label bnela addr bnelr bnectrl (Psuedo for bc 4.2.Target)	-
bng target	Branch if not greater than	bng label bnga addr bnglr bngctr bngl label bngle addr bnglr bngctrl	-
bso target	Branch if summary overflow	bso label bsoa addr bsolr bsoctr bsol label bsola addr bsolr bsocctrl	-
bns target	Branch if not summary overflow	bns label bnsa addr bnslr bnsctr bnsl label bnsle addr bnslr bnsctrl	-
bun target	Branch if unordered	bun label buna addr bunlr bunctr bunl label bunle addr bunlr bunctrl	-
bnu target	Branch if not unordered	bnu label bnua addr bnulr bnuctr bnul label bnule addr bnulr bnuctrl	-
<b>Opcode</b>	<b>Bcc PsuedoOp</b>	<b>Details a=Absolute l=to LinkRegister ctr=to CTR l=Link</b>	
bt cond,target	Branch if Condition True	bta btlr btctr btl bta btlr btctrl	-
bf cond,target	Branch if Condition False	bfa bflr bfctr bfl bfa bflr bfctrl	-
bdnz target	Branch after Decrement if nonzero	Decrement Count Register CTR, branch if CTR nonzero bdnza bdnzlr bdnzlr bdnzlr	-
bdnzt cond,target	Branch after Decrement if nonzero and condition cond TRUE	Decrement Count Register CTR, branch if CTR nonzero AND condition true (cond=LT/GT/EQ/SO/UN) bdnzt a bdnztlr bdnztl bdnztl a bdnztlr	-
bdnzf cond,target	Branch after Decrement if nonzero and condition cond FALSE	Decrement Count Register CTR, branch if CTR nonzero AND condition false (cond=LT/GT/EQ/SO/UN) bdnzfa bdnzflr bdnzfl bdnzfl a bdnzflr	-
bdz target	Branch after Decrement if zero	Decrement Count Register CTR, branch if CTR zero bdza bdzlr bdzlr bdzla bdzlr	-
bdzt cond,target	Branch after Decrement if zero and condition cond TRUE	Decrement Count Register CTR, branch if CTR zero AND condition true bdzta bdztlr bdztl bdztl a bdztlr	-
bdzf cond,target	Branch after Decrement if zero and condition cond FALSE	Decrement Count Register CTR, branch if CTR zero AND condition false bdzfa bdzflr bdzfl bdzfl a bdzflr	-
<b>Opcode</b>	<b>Condition Register Logical Instructions</b>	<b>Details</b>	
crand crbD,crbA,crbB	Condition Register AND	single bit crbA is ANDed with crbB and stored in crbD.	-
cror crbD,crbA,crbB	Condition Register OR	single bit crbA is ORed with crbB and stored in crbD.	-
crxor crbD,crbA,crbB	Condition Register XOR	single bit crbA is XORed with crbB and stored in crbD.	-
crand crbD,crbA,crbB	Condition Register NAND	single bit crbA is ANDed with crbB and the complement (NOT) is stored in crbD.	-
crnor crbD,crbA,crbB	Condition Register NOR	single bit crbA is ORed with crbB and the complement (NOT) is stored in crbD.	-
creqv crbD,crbA,crbB	Condition Register Equivalent	single bit crbA is CORed with crbB and the complement (NOT) is stored in crbD.	-
crandc crbD,crbA,crbB	Condition Register AND with Complement	single bit crbA is ANDed with the compliment (bit flipped) of crbB and stored in crbD.	-
crorc crbD,crbA,crbB	Condition Register OR with Complement	single bit crbA is ORed with the compliment (bit flipped) of crbB and stored in crbD.	-
mcrf crfD,crfS	Move Condition Register Field	The contents of crfS are copied into crfD. No other condition register fields are changed.	-
<b>Opcode</b>	<b>System Linkage Instructions</b>	<b>Details</b>	
sc	System Call		-
rfi	Return from Interrupt		-
<b>Opcode</b>	<b>Trap Instructions and Mnemonics</b>	<b>Details</b>	
twi TO,rA,SIMM	Trap Word Immediate	Trap if condition TO comparing rA to SIMM (TO=01/2/3/4=LT/GT/EQ/LogicalLT/LogicalGT) twli rA,SIMM twlei rA,SIMM tweqi rA,SIMM twgei rA,SIMM twgti rA,SIMM twnli rA,SIMM twnei rA,SIMM twliti rA,SIMM twngi rA,SIMM twlti rA,SIMM twlei rA,SIMM twgei rA,SIMM twgti rA,SIMM twnli rA,SIMM twngi rA,SIMM	-
tw TO,rA,rB	Trap Word	Trap if condition TO comparing rA to rB (TO=01/2/3/4=LT/GT/EQ/LogicalLT/LogicalGT) twlr rA,rB twle rA,rB tweq rA,rB twge rA,rB twgt rA,rB twnl rA,rB twne rA,rB twlir rA,rB twng rA,rB twllr rA,rB twlle rA,rB twlge rA,rB twlgt rA,rB twlnl rA,rB twlng rA,rB	-
<b>Opcode</b>	<b>Move to/from Machine State Register/Condition Register Instructions</b>	<b>Details</b>	
mtrcf CRM,rS	Move to Condition Register Fields	CRM=bits 0-7... eg 0xFF=all CRM fields	-
mcrxr crfD	Move to Condition Register	Move from XER bits 0-3 (SO OV CA --). Clear these bits in XER	-
mfcrr rD	Move from Condition Register		-
mtmsr rS	Move to Machine State Register		-
mfmshr rD	Move from Machine State Register		-

Opcode	Cache Management Supervisor-Level Instructions	Details	
<b>dcbi</b> rA,rB	Data Cache Block Invalidate		-
Opcode	User-Level Cache Instructions	Details	
<b>dcbt</b> rA,rB	Data Cache Block Touch		-
<b>dcbst</b> rA,rB	Data Cache Block Touch for Store		-
<b>clcs</b> rD,rA	Cache Line Compute Size		-
<b>dcbz</b> rA,rB	Data Cache Block Set to Zero		-
<b>dcbst</b> rA,rB	Data Cache Block Store		-
<b>dcbf</b> rA,rB	Data Cache Block Flush		-
Opcode	Move to/from Special Purpose Register Instructions	Details	
<b>mtspr</b> SPR,rS	Move to Special Purpose Register	Transfer rS to SPR (MQ,XER,RTCU,RTCL,DEC,LR,CTR). Simplified versions exist: <b>mtxer</b> rA <b>mtspr</b> 1,rA <b>mtlr</b> rA <b>mtspr</b> 8,rA <b>mtctr</b> rA <b>mtspr</b> 9,rA	-
<b>mfspr</b> rD,SPR	Move from Special Purpose Register	Transfer Special SPR to rD. (MQ,XER,RTCU,RTCL,DEC,LR,CTR). Simplified versions exist: <b>mfxer</b> rA <b>mfspr</b> rA,1 <b>mflr</b> rA <b>mfspr</b> rA,8 <b>mfctr</b> rA <b>mfspr</b> rA,9	-
Opcode	Segment Register Manipulation Instructions	Details	
<b>mtsr</b> SR,rS	Move to Segment Register	The contents of rS is placed into segment register specified by operand SR. (Supervisor instruction)	-
<b>mtsrin</b> rS,rB	Move to Segment Register Indirect	The contents of rS are copied to the segment register selected by bits 0–3 of rB. (Supervisor instruction)	-
<b>mfsr</b> rD,SR	Move from Segment Register	The contents of the segment register specified by operand SR are placed into rD. (Supervisor instruction)	-
<b>mfsrin</b> rD,rB	Move from Segment Register Indirect	The contents of the segment register selected by bits 0–3 of rB are copied into rD. (Supervisor instruction)	-
Opcode	Translation Lookaside Buffer Management Instruction	Details	
<b>tlbie</b> rB	Translation Lookaside Buffer Invalidate Entry		-
Opcode	External Control Instructions	Details	
<b>eciwx</b> rD,rA,rB	External Control Input Word Indexed		Y
<b>ecowx</b> rS,rA,rB	External Control Output Word Indexed		Y
Opcode	Memory Synchronization Instructions	Details	
<b>eieio</b>	Enforce In-Order Execution of I/O		-
<b>isync</b>	Instruction Synchronize		-
<b>lwarx</b> rD,rA,rB	Load Word and Reserve Indexed		-
<b>stwcx.</b> rS,rA,rB	Store Word Conditional Indexed		-
<b>sync</b>	Synchronize		-

## Floating point instructions

Opcode	Floating-Point Arithmetic Instructions	Details	.=updateCR
<b>fadd</b> frD,frA,frB	Floating-Point Add	<b>fadd.</b> frD,frA,frB	
<b>fadds</b> frD,frA,frB	Floating-Point Add Single-Precision	<b>fadds.</b> frD,frA,frB	
<b>fsub</b> frD,frA,frB	Floating-Point Subtract	<b>fsub.</b> frD,frA,frB	
<b>fsubs</b> frD,frA,frB	Floating-Point Subtract Single-Precision	<b>fsubs.</b> frD,frA,frB	
<b>fmul</b> frD,frA,frC	Floating-Point Multiply	<b>fmul.</b> frD,frA,frC	
<b>fmuls</b> frD,frA,frC	Floating-Point Multiply Single-Precision	<b>fmuls.</b> frD,frA,frC	
<b>fdiv</b> frD,frA,frB	Floating-Point Divide	<b>fdiv.</b> frD,frA,frB	
<b>fdivs</b> frD,frA,frB	Floating-Point Divide Single-Precision	<b>fdivs.</b> frD,frA,frB	
Opcode	Floating-Point Multiply-Add Instructions	Details	.=updateCR
<b>fmadd</b> frD,frA,frC,frB	Floating-Point Multiply-Add	<b>fmadd.</b> frD,frA,frC,frB	
<b>fmadds</b> frD,frA,frC,frB	Floating-Point Multiply-Add Single-Precision	<b>fmadds.</b> frD,frA,frC,frB	
<b>fmsub</b> frD,frA,frC,frB	Floating-Point Multiply-Subtract	<b>fmsub.</b> frD,frA,frC,frB	
<b>fmsubs</b> frD,frA,frC,frB	Floating-Point Multiply-Subtract Single-Precision	<b>fmsubs.</b> frD,frA,frC,frB	
<b>fnmadd</b> frD,frA,frC,frB	Floating-Point Negative Multiply-Add	<b>fnmadd.</b> frD,frA,frC,frB	
<b>fnmadds</b> frD,frA,frC,frB	Floating-Point Negative Multiply-Add Single-Precision	<b>fnmadds.</b> frD,frA,frC,frB	
<b>fnmsub</b> frD,frA,frC,frB	Floating-Point Negative Multiply-Subtract	<b>fnmsub.</b> frD,frA,frC,frB	
<b>fnmsubs</b> frD,frA,frC,frB	Floating-Point Negative Multiply-Subtract Single-Precision	<b>fnmsubs.</b> frD,frA,frC,frB	
Opcode	Floating-Point Rounding and Conversion Instructions	Details	
<b>frsp</b> frD,frB	Floating-Point Round to Single-Precision	<b>frsp.</b> frD,frB	
<b>fctiw</b> frD,frB	Floating-Point Convert to Integer Word	<b>fctiw.</b> frD,frB	
<b>fctiwz</b> frD,frB	Floating-Point Convert to Int Word with Round Toward Zero	<b>fctiwz.</b> frD,frB	
Opcode	Floating-Point Compare Instructions	Details	
<b>fcmpu</b> crfD,frA,frB	Floating-Point Compare Unordered		
<b>fcmpo</b> crfD,frA,frB	Floating-Point Compare Ordered		
Opcode	Floating-Point Compare Instructions	Details	
<b>mffs</b> frD	Move from FPSCR	<b>mffs.</b> frD	
<b>mcrfs</b> crfD,crfS	Move to Condition Register from FPSCR	<b>mtfsf.</b> crfD,IMM	
<b>mtfsfi</b> crfD,IMM	Move to FPSCR Field Immediate		
<b>mtfsf</b> FM,frB	Move to FPSCR Fields	<b>mtfsf.</b> FM,frB	
<b>mtfsb0</b> crbD	Move to FPSCR Bit 0	<b>mtfsb0.</b> crbD	
<b>mtfsb1</b> crbD	Move to FPSCR Bit 1	<b>mtfsb1.</b> crbD	

**IBM 370**

Instruction (360 - Integer)	Description	Fmt	Op	CI	Inst (360 - Float)	Description	Fmt	Op	CI	Pseudo Op	Description
A R1,D2(X2,B2)	Add	RX	5A	c	AD R1,D2(X2,B2)	Add Normalized (L)	RX	6A	c	B R1 / BR R1	Branch to Register
AR R1,R2	Add	RR	1A	c	ADR R1,R2	Add Normalized (L)	RR	2A	c	BH R1 / BHR R1	Branch on A High (after compare)
VA VR1, VR3,RS2(RT2)	Add	VST	A420	IM	AE R1,D2(X2,B2)	Add Normalized (S)	RX	7A	c	BL R1 / BLR R1	Branch on A Low (after compare)
AP D1(L1,B1),D2(L2,B2)	Add Decimal	SS	FA	c	AER R1,R2	Add Normalized (S)	RR	3A	c	BE R1 / BER R1	Branch on A Equal to B (after compare)
AH R1,D2(X2,B2)	Add Halfword	RX	4A	c	AW R1,D2(X2,B2)	Add Unnormalized (L)	RX	6E	c	BNH R1 / BNHR R1	Branch on A Not High (after compare)
ALR R1,R2	Add Logical	RR	1E	c	AWR R1,R2	Add Unnormalized (L)	RR	2E	c	BNL R1 / BNLR R1	Branch on A Not Low (after compare)
AXR R1,R2	Add Normalized (E)	RR	3E	c	AU R1,D2(X2,B2)	Add Unnormalized (S)	RX	7E	c	BNE R1 / BNER R1	Branch on A Equal to B (after compare)
AL R1,D2(X2,B2)	Add Logical	RX	5E	c	AUR R1,R2	Add Unnormalized (S)	RR	3E	c	BP R1 / BPR R1	Branch on Plus (after arithmetic)
N R1,D2(X2,B2)	AND	RX	54	c	CD R1,D2(X2,B2)	Compare (L)	RX	69	c	BM R1 / BMR R1	Branch on Minus (after arithmetic)
NC D1(L,B1),D2(B2)	AND	SS	D4	c	CDR R1,R2	Compare (L)	RR	29	c	BZ R1 / BZR R1	Branch on Zero (after arithmetic)
NI D1(B1),I2	AND	SI	94	c	CE R1,D2(X2,B2)	Compare (S)	RX	79	c	BO R1 / BOR R1	Branch on Overflow (after arithmetic)
NR R1,R2	AND	RR	14	c	CER R1,R2	Compare (S)	RR	39	c	BNP R1 / BNP R1	Branch on Not Plus (after arithmetic)
BAL R1,D2(X1,B2)	Branch and Link	RR	45	c	DD R1,D2(X2,B2)	Divide (L)	RX	5D	c	BNM R1 / BNM R1	Branch on Not Minus (after arithmetic)
BALR R1,R2	Branch and Link	RR	05	c	DDR R1,R2	Divide (L)	RR	2D	c	BNZ R1 / BNZR R1	Branch on Not Zero (after arithmetic)
BAS R1,D2(X2,B2)	Branch and Save	RR	4D	c	DE R1,D2(X2,B2)	Divide (S)	RX	7D	c	BNO R1 / BNOR R1	Branch on Not Overflow (after arithmetic)
BASR R1,R2	Branch and Save	RR	0D	c	DER R1,R2	Divide (S)	RR	3D	c	BO R1 / BOR R1	Branch if Ones (after mask test)
BC M1,D2(X2,B2)	Branch on Condition	RS	47	c	HDR R1,R2	Halve (L)	RR	24	c	BM R1 / BMR R1	Branch if Mixed (after mask test)
BCR M1,R2	Branch on Condition	RR	07	c	HER R1,R2	Halve (S)	RR	34	c	BZ R1 / BZR R1	Branch if Zero (after mask test)
BCT R1,D2(X2,B2)	Branch on Count	RX	46	c	LD R1,D2(X2,B2)	Load (L)	RX	68	c	BNO R1 / BNOR R1	Branch if Not Ones (after mask test)
BCTR R1,R2	Branch on Count	RR	06	c	LDR R1,R2	Load (L)	RR	28	c	BNM R1 / BNM R1	Branch if Not Mixed (after mask test)
BXH R1,R3,D2(B2)	Branch on Index High	RS	86	c	LE R1,D2(X2,B2)	Load (S)	RX	78	c	BNZ R1 / BNZR R1	Branch if Not Zero (after mask test)
BLXLE R1,R3,D2(B2)	Branch on Index Low	RS	87	c	LER R1,R2	Load (S)	RR	38	c	NOP	No Operation
C R1,D2(X2,B2)	Compare	RX	59	c	LTDR R1,R2	Load and Test (L)	RR	22	c	WTO Text'	Write To Operator
CP D1(L1,B1),D2(L2,B2)	Compare Decimal	SS	F9	c	LTER R1,R2	Load and Test (S)	RR	32	c		
CH R1,D2(X2,B2)	Compare Halfword	RX	49	c	LCDR R1,R2	Load Complement (L)	RR	23	c		
CL R1,D2(X2,B2)	Compare Logical	RX	55	c	LCER R1,R2	Load Complement (S)	RR	33	c		
CLC D1(L,B1),D2(B2)	Compare Logical	SS	D5	c	LNDR R1,R2	Load Negative (L)	RR	21	c		
CLI D1(B1),I2	Compare Logical	SI	95	c	LNER R1,R2	Load Negative (S)	RR	31	c		
CVB R1,D2(X2,B2)	Convert to Binary	RX	4F	c	LPDR R1,R2	Load Positive (L)	RR	20	c		
CVD R1,D2(X2,B2)	Convert to Decimal	RX	4E	c	LPDR R1,R2	Load Positive (L)	RR	30	c		
D R1,D2(X2,B2)	Divide	RX	5D	c	LRDR R1,R2	Load Rounded (E/L)	RR	25	c		
DR R1,R2	Divide	RR	1D	c	LRER R1,R2	Load Rounded (L/S)	RR	35	c		
DP D1(L1,B1),D2(L2,B2)	Divide Decimal	SS	FD	c	MXR R1,R2	Multiply (E)	RR	26	c		
ED D1(L1,B1),D2(B2)	Edit	SS	DE	c	MD R1,D2(X2,B2)	Multiply (L)	RR	6C	c		
EDMK D1(L,B1),D2(B2)	Edit and Mark	SS	DF	c	MDR R1,R2	Multiply (L)	RR	2C	c		
X R1,D2(X2,B2)	Exclusive OR	RX	57	C	MXD R1,D2(X2,B2)	Multiply (L/E)	RR	67	c		
XC D1(L,B1),D2(B2)	Exclusive OR	SS	D7	C	MXDR R1,R2	Multiply (L/E)	RR	27	c		
XI D1(B1),I2	Exclusive OR	SI	97	C	ME R1,D2(X2,B2)	Multiply (S/L)	RX	7C	c		
XR R1,R2	Exclusive OR	RR	17	C	MER R1,R2	Multiply (S/L)	RR	3C	c		
EX R1,D2(X2,B2)	Execute	RX	44	c	STD R1,D2(X2,B2)	Store (L)	RX	60	c		
HIO D2(B2)	Halt I/O	S	9E00	pc	STE R1,D2(X2,B2)	Store (S)	RR	70	c		
IC R1,D2(X2,B2)	Insert Character	RX	43	c	SXR R1,R2	Subtract Normalized (E)	RR	37	c		
ISK R1,R2	Insert Storage Key	RR	09	p	SD R1,D2(X2,B2)	Subtract Normalized (L)	RR	6B	c		
L R1,D2(X2,B2)	Load	RX	58	c	SDR R1,R2	Subtract Normalized (L)	RR	2B	c		
LR R1,R2	Load	RR	18	c	SE R1,D2(X2,B2)	Subtract Normalized (S)	RX	7B	c		
LA R1,D2(X2,B2)	Load Address	RX	41	c	SER R1,R2	Subtract Normalized (S)	RR	3B	c		
LTR R1,R2	Load and Test	RR	12	c	SW R1,D2(X2,B2)	Subtract Unnormalized (L)	RX	6F	c		
LCR R1,R2	Load Complement	RR	13	c	SWR R1,R2	Subtract Unnormalized (L)	RR	2F	c		
LH R1,D2(X2,B2)	Load Halfword	RX	48	c	SU R1,D2(X2,B2)	Subtract Unnormalized (S)	RX	7F	c		
LM R1,R2,D2(B2)	Load Multiple (regs R1-R2)	RS	98	c	SUR R1,R2	Subtract Unnormalized (S)	RR	3F	c		
LNR R1,R2	Load Negative	RR	11	c							
LPR R1,R2	Load Positive	RR	10	c							
LPSW D2(B2)	Load PSW	S	82	pn							
LRA R1,D2(X2,B2)	Load Real Address	RX	B1	pc							
MVC D1(L,B1),D2(B2)	Move Characters	SS	D2	c							
MVI D1(B1),I2	Move Immediate	SI	92	c							
MVN D1(L,B1),D2(B2)	Move Numerics	SS	D1	c							
MVO D1(L1,B1),D2(L2,B2)	Move with Offset	SS	F1	c							
MVZ D1(L,B1),D2(B2)	Move Zones	SS	D3	c							
M R1,D2(X2,B2)	Multiply	RX	5C	c							
MR R1,R2	Multiply	RR	1C	c							
MP D1(L1,B1),D2(L2,B2)	Multiply Decimal	SS	FC	c							
MH R1,D2(X2,B2)	Multiply Halfword	RX	4C	c							
O R1,D2(X2,B2)	OR	RX	56	c							
OC D1(L,B1),D2(B2)	OR	SS	D6	c							
OI D1(B1),I2	OR	SI	96	c							
OR R1,R2	OR	RR	16	c							
PACK D1(L1,B1),D2(L2,B2)	Pack	SS	F2	c							
RDD D1(B1),I2	Read Direct	SI	85	p							
SPM R1	Set Program Mask	RR	04	n							
SSK R1,R2	Set Storage Key	RR	08	p							
SSM D2(B2)	Set System Mask	S	80	p							
SLDA R1,D2(B2)	Shift Left Double	RS	8F	c							
SLDL R1,D2(B2)	Shift Left Double Logical	RS	8D	c							
SLA R1,D2(B2)	Shift Left Single	RS	8B	c							
SLL R1,D2(B2)	Shift Left Single Logical	RS	89	c							
SRDA R1,D2(B2)	Shift Right Double	RS	8E	c							
SRDL R1,D2(B2)	Shift Right Double Logical	RS	8C	c							
SRA R1,D2(B2)	Shift Right Single	RS	8A	c							
SRL R1,D2(B2)	Shift Right Single Logical	RS	88	c							
SIO D2(B2)	Start I/O	S	9C00	pc							
ST R1,D2(X2,B2)	Store	RX	50	c							
STC R1,D2(X2,B2)	Store Character	RX	42	c							
STH R1,D2(X2,B2)	Store Halfword	RX	40	c							
STM R1,R3,D2(B2)	Store Multiple (regs R1-R2)	RS	90	c							
S R1,D2(X2,B2)	Subtract	RX	5B	c							
SP D1(L1,B1),D2(L2,B2)	Subtract Decimal	SS	FB	c							
SH R1,D2(X2,B2)	Subtract Halfword	RX	4B	c							
SL R1,D2(X2,B2)	Subtract Logical	RX	5F	c							
SLR R1,R2	Subtract Logical	RR	1F	c							
SVC I	Supervisor Call	RR	0A	c							
TS D2(B2)	Test and Set	S	93	c							
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							
TR D1(L,B1),D2(B2)	Translate	SS	DC	c							
TRT D1(L,B1),D2(B2)	Translate and Test	SS	DD	c							
UNPK D1(L1,B1),D2(L2,B2)	Unpack	SS	F3	c							
ZAP D1(L1,B1),D2(L2,B2)	Zero and Add	SS	F8	C							

**Assembler Instruction Description**

DC	Define constant
DS	Define storage
CCW	Define channel command word
ccwo--	Define format-0 channel command word
CCW1**	Define format-1 channel command word
START	Start assembly
LOCTR**	Specify multiple location counters
CSE CT	Identify control section
DSECT	Identify dummy section
DXD*	Define external dummy section
CXD*	Cumulative length of external dummy section
COM	Identify blank common control section
AMODE**	Specify addressing mode
RMODE**	Specify residence mode
ENTRY	Identify entry-point symbol
EXTRN	Identify external symbol
WXTRN	Identify weak external symbol
USING	Use base address register
DROP	Drop base address register
TITLE	Identify assembly output
EJECT	Start new page
SPACE	Space listing
PRINT	Print optional J data
ICTL	Input format control
ISEQ	Input sequence checking
PUNCH	Punch a card
RE PRO	Reproduce following card
ORG	Set location counter
EQU	Equate symbol
OPSYN*	Equate operation code
PUSH*	Save current PRINT or USING status
POP*	Restore PRINT or USING status
LTORG	Begin literal pool
CNOP	Conditional no operation
COPLY	Copy predefined source coding
END	End assembly
MACRO	Macro definition header
MEXIT	Macro definition exit
MEND	Macro definition trailer
AREAD**	Assign card to SETC symbol
ACTR	Conditional assembly loop counter
AGO	Unconditional branch
AIF	Conditional branch
ANOP	Assembly no operation
GBLA	Define global SETA symbol
GBLB	Define global SETB symbol
GBLC	Define global SETC symbol
LCLA	Define local SET A symbol
LCLB	Define local SETB symbol
LCLC	Define local SETC symbol
MNOTE	Generate error message
MHELP**	Trace macro flow
SETA	Set arithmetic variable symbol
SETB	Set binary variable symbol
SETC	Set character variable symbol

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.  
(L/E) 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; (VCT - 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.

1, 2, 3: Denotes association with first, second, or third operand  
B1, B2: Base register designation field  
D1, D2: Displacement field (12 bit)  
GR2: Register designation field (general register)  
I2: Immediate operand field (8 bit)  
L: Length field (8 bit)  
L1, L2: Length field (4 bit)  
QR3: Register designation field (equivalent to GR3 if general register, or FR3 if floating-point register)  
R1, R2, R3: Register designation field  
RS2: Register designation field (starting address of vector)  
RT2: Register designation field (stride of vector)  
VR1, VR2, VR3: Register designation field (vector register)  
X2: Index register designation field

Instruction (370)	Description	Fmt	Op	CI	Instruction	Description	Fmt	Op	CI
VACD VR1,RS2(RT2)	Accumulate (L)	VST	A417	IM	VLZER VR1	Load Zero (S)	VV	A501C	
VACDR VR1,VR2	Accumulate (L)	VV	A517	IM	VMXAD VR1,FR3,GR2	Maximum Absolute (L)			

VACER VR1,VR2	Accumulate (S/L)	VV	A507	IM	VMXSD VR1,FR3,GR2Maximum Signed (L)	VR	A610IM
VAQ VR1,GR3,VR2	Add	QV	A5A0	IM	VMXSE VR1,FR3,GR2Maximum Signed (S)	VR	A600IM
VAR VR1,VR3,VR2	Add	VV	A520	IM	VMNSD VR1,FR3,GR2Minimum Signed (L)	VR	A611IM
VAS VR1,GR3,RS2(RT2)	Add	QST	A4A0	IM	VMNSE VR1,FR3,GR2Minimum Signed (S)	VR	A601IM
VADQ VR1,FR3,VR2	Add (L)	QV	A490	IM	MC D1(B1),I2	Monitor Call	SI AF
VADR VR1,VR3,VR2	Add (L)	VV	A510	IM	MVCIN D1(L,B1),D2(B)Move Inverse	SS	E8
VADS VR1,FR3,RS2(RT2)	Add (L)	QST	A490	IM	MVCL R1,R2	Move Long	RR 0E i c
VAD VR1,VR3,RS2(RT2)	Add (L)	VST	A410	IM	MVCP D1(R1,B1),D2(I)Move to Primary	SS	DA qc
VAE VR1,VR3,RS2(RT2)	Add (S)	VST	A400	IM	MVCS D1(R1,B1),D2(I)Move to Secondary	SS	DB qc
VAEQ VR1,FR3,VR2	Add (S)	QV	A580	IM	MVCK D1(R1,B1),D2(I)Move with Key	SS	D9 qc
VAER VR1, VR3,VR2	Add (S)	VV	A500	IM	VMQ VR1,GR3,VR2	Multiply	QV A5A2IM
VAES VR1,FR3,RS2(RT2)	Add (S)	QST	A480	IM	VMR VR1,VR3,VR2	Multiply	VV A522IM
VN VR1,VR3,RS2(RT2)	AND	VST	A424	IM	VMS VR1,GR3,RS2(R)Multiply	QST	A5A2IM
VNR VR1,GR3,VR2	AND	QV	A5A4	IM	VM VR1,VR3,RS2(RT)Multiply	VST	A522IM
VNS VR1,GR3,RS2(RT2)	AND	VV	A524	IM	VMD VR1,VR3,RS2(R)Multiply (L)	VST	A412IM
VNVM RS2	AND to VMR	QST	A4A4	IM	VMDQ VR1,FR3,VR2	Multiply (L)	QV A592IM
CLRCH D2(B2)	Clear Channel	S	9F01	pc	VMDR VR1,VR3,VR2	Multiply (L)	VV A512IM
CLRIO D2(B2)	Clear I/O	S	9D01	pc	VMDS VR1,FR3,RS2(I)Multiply (L)	QST	A492IM
VRCL D2(B2)	Clear VR	S	A6C5	IZ	VME VR1,VR3,RS2(R)Multiply (S/L)	VST	A402IM
CLM R1,M3,D2(B2)	Comp Logical Chars under	RS	BD	c	VMEQ VR1,FR3,VR2	Multiply (S/L)	QV A582IM
VC M1,VR3,RS2(RT2)	Compare	VST	A428	IC	VMER VR1,VR3,VR2	Multiply (S/L)	VV A502IM
CR R1,R2	Compare	RR	19	c	VMES VR1,FR3,RS2(I)Multiply (S/L)	QST	A482IM
VCQ M1,GR3,VR2	Compare	QV	A5A8	IC	VMCD VR1,VR3,RS2(I)Multiply and Accumulate (L)	VST	A416IM
VCR M1,VR3,VR2	Compare	VV	A528	IC	VMCDR VR1,VR3,VR2Multiply and Accumulate (L)	VV	A516IM
VCS M1,GR3,RS2(RT2)	Compare	QST	A4A8	IC	VMCE VR1,VR3,RS2(I)Multiply and Accumulate (S/L)VST	A406IM	
VCD M1,VR3,RS2(RT2)	Compare (L)	VST	A418	IC	VMCER VR1,VR3,VR2Multiply and Accumulate (S/L)VV	A506IM	
VCDQ M1,FR3,VR2	Compare (L)	QV	A598	IC	VMAD VR1,VR3,RS2(I)Multiply and Add (L)	VST	A414IM
VCDR M1,VR3,VR2	Compare (L)	VV	A518	IC	VMADQ VR1,FR3,VR2Multiply and Add (L)	QV	A594IM
VCD S M1,FR3,RS2(RT2)	Compare (L)	QST	A498	IC	VMADS VR1,FR3,RS2Multiply and Add (L)	QST	A494IM
VCE M1,VR3,RS2(RT2)	Compare (S)	VST	A408	IC	VMAE VR1,VR3,RS2(I)Multiply and Add (S/L)	VST	A404IM
VCEQ M1,FR3,VR2	Compare (S)	QV	A588	IC	VMAEQ VR1,FR3,VR2Multiply and Add (S/L)	QV	A584IM
VCE R M1,VR3,VR2	Compare (S)	VV	A508	IC	VMAES VR1,FR3,RS2Multiply and Add (S/L)	QST	A484IM
VCES M1,FR3,RS2(RT2)	Compare (S)	QST	A488	IC	VMSD VR1,VR3,RS2(I)Multiply and Subtract (L)	VST	A415IM
CS R1,R3,D2,(B2)	Compare and Swap	RS	BA	c	VMSDQ VR1,FR3,VR2Multiply and Subtract (L)	QV	A595IM
CDS R1,R3,D2(B2)	Compare Double and Swap	RS	BB	c	VMSDS VR1,FR3,RS2Multiply and Subtract (L)	QST	A495IM
CLR R1,R2	Compare Logical	RR	15	c	VMSE VR1,VR3,RS2(I)Multiply and Subtract (S/L)	VST	A405IM
CLCL R1,R2	Compare Logical Long	RR	0F	i c	VMSEQ VR1,FR3,VR2Multiply and Subtract (S/L)	QV	A585IM
VCVM	Complement VMR	RRE	A641	NC	VMSER VR1,FR3,RS2Multiply and Subtract (S/L)	QST	A485IM
CONCS D2(B2)	Connect Channel Set	S	B200	pc	VO VR1,VR3,RS2(RT)OR	VST	A425IM
VCZVM GR1	Count Left Zeros in VMR	RRE	A642	NC C	BOQ VR1,GR3,VR2	OR	QV A5A1IM
VCOVM GR1	Count Ones in VMR	RRE	A643	NC C	VOR VR1,VR3,VR2	OR	VV A525IM
Model-dependent	Diagnose	-	83	PY	VOS VR1,GR3,RS2(R)OR	QST	A5A1IM
DISCS D2(B2)	Disconnect Channel Set	S	B201	pc	VOVM RS2	OR to VMR	VS A685NC
VDD VR1, VR3,RS2(RT2)	Divide (L)	VST	A413	IM	PC D2(B2)	Program Call	S B218q
VDDQ VR1,FR3,VR2	Divide (L)	QV	A493	IM	PT R1,R2	Program Transfer	RRE B228q
VDDR VR1,VR3,VR2	Divide (L)	VV	A513	IM	PTLB	Purge TLB	S B201p
VDDS VR1,FR3,RS2(RT2)	Divide (L)	QST	A493	IM	RRB D2(B2)	Reset Reference Bit	S B213pc
VDE VR1,VR3,RS2(RT2)	Divide (S)	VST	A403	IM	RABE R1,R2	Reset Reference Bit Extend	RRE B224pc
VDEQ VR1,FR3,VR2	Divide (S)	QV	A583	IM	VACRS D2(B2)	Restore VAC	S A6C1NO P
VDER VR1,VR3,VR2	Divide (S)	VV	A503	IM	VMRRS D2(B2)	Restore VMR	S A6C3NZ
VDES VR1,FR3,RS2(RT2)	Divide (S)	QST	A483	IM	VRRS GR1	Restore VR	RRE A648I Z XC
VX VR1, VR3,RS2(RT2)	Exclusive OR	VST	A426	IM	VSRRS D2(B2)	Restore VSR	S A6C2I Z X
VXQ VR1,GR3,VR2	Exclusive OR	QV	A5A6	IM	RIO D2(B2)	Resume I/O	S 9C02pc
VXR VR1,VR3,VR2	Exclusive OR	VV	A526	IM	VRSVC GR1	Save Changed VR	RRE A649I Z PC
VXS VR1,GR3,RS2(RT2)	Exclusive OR	QST	A4A6	IM	VACSV D2(B2)	Save VAC	S A6C1NO P
VXVM RS2	Exclusive OR to VMR	VS	A686	NC	VMRSV D2(B2)	Save VMR	S A6C1NZ
VXEL VR1,GR3,GR2	Extract Element	VR	A629	N1	VRSV GR1	Save VR	RRE A641I Z C
VXELD VR1,FR3,GR2	Extract Element (L)	VR	A619	N1	VS RSV D2(B2)	Save VSR	S A6C1NO X
VXELE VR1,FR3,GR2	Extract Element (S)	VR	A609	N1	SAC D2(B2)	Set Address Space Control	S B219q
EPAR R1	Extract Primary ASN	RRE	B226	q	SCK D2(B2)	Set Clock	S B204pc
ESAR R1	Extract Secondary ASN	RRE	B227	q	SCKC D2(B2)	Set Clock Comparator	S B206p
VXVC GR1	Extract VCT	RRE	A644	NO	SPT D2,(B2)	Set CPU Timer	S B208p
VXVMM GR1	Extract Vector Mask Mode	RRE	A646	NO	SPX D2,(B2)	Set Prefix	S B210p
HDV D2(B2)	Halt Device	S	9E01	pc	SPKA D2(B2)	Set PSW Key from Address	S B20Aq
IAC R1	Insert Address Space Contr	RRE	B224	qp	SSAR R1	Set Secondary ASN	RRE B225q
ICM R1,M3,D2(B2)	Insert Characters under Mas	RS	BS	c	SSKE R1,R2	Set Storage Key Extended	RRE B22Ep
IPK	Insert PSW Key	S	B20B	q	VSVMM D2(B2)	Set Vector Mask Mode	S A6C1NO
ISKE R1,R2	Insert Storage Key Extended	RRE	B229	p	SRP D1(L,B1),D2(B2)Shift and Round Decimal	SS	F0 c
IVSK R1,R2	Insert Virtual Storage Key	RRE	B223	q	VSLL VR1, VR3,D2(B)Shift Left Single Logical	RSE	E425IM
IPTE R1,R2	Invalidate Page Table Entry	RRE	B221	p	VSRL VR1,VR3,D2(B)Shift Right Single Logical	RSE	E424IM
VLR VR1,VR2	Load	VV	A509	IC	SIGP R1,R3,D2(B2)	Signal Processor	RS AE pc
VLQ VR1,GR2	Load	QV	A5A9	IC	SIOF D2(B2)	Start I/O Fast Release	S 9C01pc
VLD VR1,RS2(RT2)	Load (L)	VST	A419	IC	VST VR1,RS2(RT2)	Store	VST A401IC
VLDO VR1,FR2	Load (L)	QV	A599	IC	VSTD VR1,RS2(RT2)	Store (L)	VST A411IC
VLDR VR1,VR2	Load (L)	VV	A519	IC	VSTE VR1,RS2(RT2)	Store (S)	VST A401IC
VLE VR1,RS2(RT2)	Load (S)	VST	A409	IC	STIDC D2,(B2)	Store Channel ID	S B203pc
VLEQ VR1,FR2	Load (S)	QV	A608	IC	STCM R1,M3,D2(B2)	Store Characters under Mas	RS BE
VLER VR1,VR2	Load (S)	VV	A589	IC	STCK D2,(B2)	Store Clock	S B205c
LASP D1(B1),D2(B2)	Load Address Space Param	SSE	E500	pc	STCKC D2,(B2)	Store Clock Comparator	S B207p
VLBX VR1,GR3,D2(B2)	Load Bit Index	RSE	E428	IG C	VSTK VR1,RS2(RT2)	Store Compressed	VST A40FIC
VLCR VR1,VR2	Load Complement	VV	A562	IM	VSTKD VR1,RS2(RT2)Store Compressed (L)	VST	A41FIC
VLCDR VR1,VR2	Load Complement (L)	VV	A552	IM	VSTKE VR1,RS2(RT2)Store Compressed (S)	VST	A40FIC
VLCER VR1,VR2	Load Complement (S)	VV	A542	IM	STCTL R1,R3,D2(B2)	Store Control	TS B6 p
LCTL R1,R3,D2(B2)	Load Control	RS	B7	p	STAP D2,(B2)	Store CPU Address	S B212p
VLEL VR1,GR3,GR2	Load Element	VR	A628	N1	STIDP D2,(B2)	Store CPU ID	S B202p
VLELD VR1,FR3,GR2	Load Element (L)	VR	A618	N1	STPT D2,(B2)	Store CPU Timer	S B209p
VLELE VR1,FR3,GR2	Load Element (S)	VR	S608	N1	VSTH VR1,RS2(RT2)	Store Halfword	VST A421IC
VLY VR1,RS2(RT2)	Load Expanded	VST	A40B	IC	VSTI VR1,VR3,D2(B2)Store Indirect	RSE	E401IC
VLYD VR1,RS2(RT2)	Load Expanded (L)	VST	A41B	IC	VSTID VR1,VR3,D2(B)Store Indirect (L)	RSE	R411IC
VLYE VR1,RS2(RT2)	Load Expanded (S)	VST	A40B	IC	VSTIE VR1,VR3,D2(B)Store Indirect (S)	RSE	R401IC
VLH VR1,RS2(RT2)	Load Halfword	VST	A429	IC	VSTM VR1,RS2(RT2)	Store Matched	VST A40EIC
VLI VR1, VR3,D2(B2)	Load Indirect	RSE	E400	IC	VSTMD VR1,RS2(RT2)Store Matched (L)	VST	A41E1C
VLID VR1,VR3,D2(B2)	Load Indirect (L)	RSE	E410	IC	VSTME VR1,RS2(RT2)Store Matched (S)	VST	A40E1C
VLIIE VR1, VR3,D2(B2)	Load Indirect (S)	RSE	E400	IC	STPX D2,(B2)	Store Prefix	S B211p
VLIINT VR1,RS2(RT2)	Load Integer Vector	VST	A42A	IC	STNSM D1(B1),I2	Store Then AND System Mas	SI AC p
VLM VR1,RS2(RT2)	Load Matched	VST	A40A	IC	STOSM D1(B1),I2	Store Then OR System Mas	SI AD p
VLMQ VR1,GR2	Load Matched	QV	A5AA	IC	VSTVP D2(B2)	Store Vector Parameters	S A6C1NO
VLMR VR1,VR2	Load Matched	VV	A50A	IC	VSTVM RS2	Store VMR	VS A682NC
VLMD VR1,RS2(RT2)	Load Matched (L)	VST	A41A	IC	SR R1,R2	Subtract	RR 1B c
VLMDQ VR1,FR2	Load Matched (L)	QV	A59A	IC	VSQ VR1,GR3,VR2	Subtract	QV A5A1IM
VLMDR VR1,VR2	Load Matched (L)	VV	A51A	IC	VSR VR1,VR3,VR2	Subtract	VV A521IM
VLME VR1,RS2(RT2)	Load Matched (S)	VST	A40A	IC	VS VR1,VR3,RS2(RT)zSubtract	VST	A421IM
VLMEQ VR1,FR2	Load Matched (S)	QV	A58A	IC	VSS VR1,GR3,RS2(R)Subtract	QST	A4A1IM
VLMER VR1,VR2	Load Matched (S)	VV	A50A	IC	VSD VR1,VR3,RS2(R)Subtract (L)	VST	A411IM
VLNR VR1,VR2	Load Negative	VV	A561	IM	VSDQ VR1,FR3,VR2	Subtract (L)	QV A591IM
VLNDR VR1,VR2	Load Negative (L)	VV	A551	IM	VSDR VR1,VR3,VR2	Subtract (L)	VV A511IM
VLNER VR1,VR2	Load Negative (S)	VV	A541	IM	VSDS VR1,FR3,RS2(F)Subtract (L)	QST	A491IM
VLPR VR1,VR2	Load Positive	VV	A560	IM	VSE VR1,VR3,RS2(R)Subtract (S)	VST	A401IM
VLPDR VR1,VR2	Load Positive (L)	VV	A550	IM	VSEQ VR1,FR3,VR2	Subtract (S)	QV A581IM
VLPDR VR1,VR2	Load Positive (S)	VV	A540	IM	VSER VR1,VR3,VR2	Subtract (S)	VV A501IM
VLVCU GR1	Load VCT and Update	RRE	A645	NO C	VSES VR1,FR3,RS2(F)Subtract (S)	QST	A481IM
VLVCA D2(B2)	Load VCT from Address	S	A6C4	NO C	VSPSD VR1,FR2	Sum Partial Sums (L)	VR A611p
VLMV RS2	Load VMR	VS	A680	NC	TB R1,R2	Test Block	RRE B221Cpc
VLCVM RS2	Load VMR Complement	VS	A681	NC	TPROT D1(B1),D2(B2)Test Protection	SSE	E501pc
VL VR1,RS2(RT2)	Load VST	VST	A409	IC	VTVM	Test VMR	RRE A640NC C
VLZR VR1	Load Zero	VV	A50B	IC	WRD D1(B1),I2	Write Direct	SI 84 P
VLZDR VR1	Load Zero (L)	VV	A51B	IC	VZPSO VR1	Zero Partial Sums (L)	VR A61EIP





# IBM Bit order

	Most Significant Bts																Least Significant Bits															
Normal	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IBM order	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Bit Value	2G	1G	512M	256M	128M	64M	32M	16M	8M	4M	2M	1M	512K	256K	128K	64K	32K	16K	8K	4K	2K	1K	512	256	128	64	32	16	8	4	2	1

Bit order on the PowerPC and IBM370 are the opposite of most systems!

EBDIC																
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x	NUL	SOH	STX	ETX	SEL	HT	RNL	DEL	GE	SPS	RPT	VT	FF	CR	SO	SI
1x	DLE	DC1	DC2	DC3	ES/EN	NL	BS	POC	CAN	EM	UBS	CU1	IFS	IGS	IRS	US/ITB
2x	DS	SOS	FS	WUS	YP/IN	LF	ETB	ESC	SA	SFE	SM/SW	CSP	MFA	ENQ	ACK	BEL
3x			SYN	IR	PP	TRN	NBS	EOT	SBS	IT	RFF	CU3	DC4	NAK		SUB
4x	SP										¢	.	<	(	+	
5x	&										!	\$	*	)	;	¬
6x	-	/									!	,	%	'	>	?
7x											:	#	@	'	=	"
8x		a	b	c	d	e	f	g	h	i						±
9x		j	k	l	m	n	o	p	q	r						
Ax		~	s	t	u	v	w	x	y	z						
Bx	^										[	]				
Cx	{	A	B	C	D	E	F	G	H	I						
Dx	}	J	K	L	M	N	O	P	Q	R						
Ex	\		S	T	U	V	W	X	Y	Z						
Fx	0	1	2	3	4	5	6	7	8	9						EO

- 0 63
- 1 62
- 2 61
- 4 59
- 5 58
- 6 57
- 7 56
- 8 55
- 11 52
- 12 51
- 15 48
- 16 47
- 17 46
- 18 45
- 19 44
- 20 43
- 23 40
- 24 39
- 31 32
- 32 31
- 33 30
- 63 0

Super-H Opcode	Instruction	Description	Function	Code	Op1 Size	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	1	ADD R0,R1
ADD #imm,Rn	ADD Binary	Adds 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	0111nnnnllllllllll	-	1	1	ADD #H01,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	-	Ovfl	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	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	1100100111111111	-	1	1	AND #H0F,R0
ANDB #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)	1100110111111111	-	3	1	ANDB #H50,@(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	10001011ddddddddd	-	3/1	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	10001111ddddddddd	Y	2/1	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	1010ddddddddd	Y	2	1	BRA TRGET
BRAF @Rn	Branch Far	Branches unconditionally. The branch destination is PC + the 32-bit contents of the general register Rn.	Rn + PC → PC	0000mmmm00100011	Y	2	1	BRAF TRGET
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	1011ddddddddd	Y	2	1	BSR TRGET
BSRF @Rn	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, Rn + PC → PC	0000mmmm00000011	Y	2	1	BSRF 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	10001001ddddddddd	-	3/1	1	BT TRGET_T
BTS label	Branch if True with Delay Slot	Reads the T bit and conditionally branches. If T = 1, BTS branches after the following instruction executes. If T = 0, BTS 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	10001101ddddddddd	Y	2/1	1	BTS TARGET_T
CLRMAC	Clear MAC Register	Clears the MACH and MACL Register.	0 → MACH, MACL	0000000001010000	-	1	1	CLRMAC
CLRT	Clear T Bit	Clears the T bit.	0 → T	0000000000010000	-	1	1	CLRT
CMPEO Rm,Rn	Compare Equal	If Rn == Rm, T = 1	When Rn = Rm, T = 1	0011nnnnmmmm0000	result	1	1	CMPEO R0,R1
CMPIGE Rm,Rn	Compare Greater or Equal (signed)	If Rn >= Rm with signed data, T = 1	When signed and Rn > Rm, 1 → T	0011nnnnmmmm0011	result	1	1	CMPIGE R0,R1
CMPIGT Rm,Rn	Compare Greater Than (Signed)	If Rn > Rm with signed data, T = 1	When signed and Rn > Rm, 1 → T	0011nnnnmmmm0111	result	1	1	CMPIGT R0,R1
CMPIHI Rm,Rn	Compare Higher (Unsigned)	If Rn > Rm with unsigned data, T = 1	When unsigned and Rn > Rm, 1 → T	0011nnnnmmmm0110	result	1	1	CMPIHI R0,R1
CMPIHS Rm,Rn	Compare Higher or Same (Unsigned)	If Rn >= Rm with unsigned data, T = 1	When unsigned and Rn > Rm, 1 → T	0011nnnnmmmm0100	result	1	1	CMPIHS R0,R1
CMPIPL Rn	Compare if Plus	If Rn > 0, T = 1	When Rn > 0, 1 → T	0100nnnn00010101	result	1	1	CMPIPL R0
CMPIZ Rn	Compare if Plus or Zero	If Rn >= 0, T = 1	When Rn >= 0, 1 → T	0100nnnn00010001	result	1	1	CMPIZ R0
CMPISTR Rm,Rn	Compare String	If a byte in Rn equals a byte in Rm, T = 1 (Any of 4 bytes, must be same position in Rm+Rn)	When byte in Rn = byte in Rm, 1 → T	0010nnnnmmmm1100	result	1	1	CMPISTR R2,R3
CMPEQ #imm,R0	Compare Equal Immediate	If R0 = imm, T = 1	When R0 = imm, 1 → T	1000100011111111	result	1	1	CMPEQ #H0F,R0
DIVOS Rm,Rn	Divide Step 0 as Signed	DIVOS is an initialization instruction for signed division. It finds the quotient by repeatedly dividing in combination with the DIVI or another instruction that divides for each bit after this instruction.	MSB of Rn → Q; MSB of Rm → M,M*Q → T	0010nnnnmmmm0111	result	1	1	DIVOS R0,R1
DIVOU	Divide Step 0 as Unsigned	DIVOU is an initialization instruction for unsigned division. It finds the quotient by repeatedly dividing in combination with the DIVI or another instruction that divides for each bit after this instruction.	0 → M/Q/T	0000000000011001	0	1	1	DIVOU
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 bit 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	result	1	1	DIV1 R0,R1
DMULS.L Rm,Rn	Double-Length Multiply as Signed	Performs Signed multiplication of 32-bit Rn and Rm, and stores the 64-bit results in MACH,MACL	With sign Rn × Rm → MACH, MACL	0011nnnnmmmm1101	-	2-4	1	DMULS.L R0,R1
DMULU.L Rm,Rn	Double-Length Multiply as Unsigned	Performs Unsigned multiplication of 32-bit Rn and Rm, and stores the 64-bit results in MACH,MACL	Without sign Rn × Rm → MACH, MACL	0011nnnnmmmm0101	-	2-4	1	DMULU.L R0,R1
DT Rn	Decrement and Test (DUNZ)	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	result	1	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	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	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	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	1	EXTU.W R0,R1
JMP @Rn	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 Rn.	Rn → PC	0100mmmm00101011	Y	2	1	JMP @R0
JSR @Rn	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 Rn. The stored saved PC is the address four bytes after this instruction.	PC → PR, Rn → PC	0100mmmm00001011	Y	2	1	JSR @R0
LDC Rm,SR	Load to Control Register	Store the source operand into control register	Rm → SR	0100mmmm00001110	LSB	1	1	LDC R0,SR
LDC Rm,GBR	Load to Control Register	Store the source operand into control register	Rm → GBR	0100mmmm00011110	-	1	1	LDC R0,GBR
LDC Rm,VBR	Load to Control Register	Store the source operand into control register	Rm → VBR	0100mmmm00010111	-	1	1	LDC R0,VBR
LDC.L @Rm+,SR	Load to Control Register	Store the source operand into control register	(Rm) → SR, Rm + 4 → Rm	0100mmmm00000111	LSB	3	1	LDC.L @R15+,SR
LDC.L @Rm+,GBR	Load to Control Register	Store the source operand into control register	(Rm) → GBR, Rm + 4 → Rm	0100mmmm00010111	-	3	1	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	1	LDC.L @R15+,VBR
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 A0C.	Rm → MACH	0100mmmm00001010	-	1	1	LDS R0,MACH
LDS Rm,MACL	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 A0C.	Rm → MACL	0100mmmm00010110	-	1	1	LDS R0,MACL
LDS Rm,PR	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 A0C.	Rm → PR	0100mmmm00101010	-	1	1	LDS R0,PR
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 A0C.	(Rm) → MACH, Rm + 4 → Rm	0100mmmm00000110	-	1	1	LDS.L @R15+,MACH
LDS.L @Rm+,MACL	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 A0C.	(Rm) → MACL, Rm + 4 → Rm	0100mmmm00010110	-	1	1	LDS.L @R15+,MACL
LDS.L @Rm+,PR	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 A0C.	(Rm) → PR, Rm + 4 → Rm	0100mmmm00100110	-	1	1	LDS.L @R15+,PR
MACL.@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 (Rm) × (Rn) → MAC → MAC	0000nnnnmmmm1111	3/2-4	1	1	MACL.@R0+,@R1+
MAC.W @Rm+,@Rn+	Multiply and Accumulate Calculation Word	Performs 16-bit multiplication of signed @Rm and @Rn, with post increment, and adds the 32-bit result in MACL. MACH is any overflow (1 bit)	With sign, (Rm) × (Rn) + MAC → MAC	0100nnnnmmmm1111	-	3/2-4	1	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	1	MOV R0,R1
MOV.B 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)	0000nnnnmmmm0000	-	1	1	MOV.B R0,@R1
MOV.W 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)	0010nnnnmmmm0001	-	1	1	MOV.W R0,@R1
MOV.L 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)	0000nnnnmmmm0010	-	1	1	MOV.L R0,@R1
MOV.B @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) → sign extension → Rn	0110nnnnmmmm0000	-	1	1	MOV.B @R0,R1
MOV.W @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) → sign extension → Rn	0110nnnnmmmm0001	-	1	1	MOV.W @R0,R1
MOV.L @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	0110nnnnmmmm0010	-	1	1	MOV.L @R0,R1
MOV.B 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.	Rn - 1 → Rn, Rm → (Rn)	0010nnnnmmmm0100	-	1	1	MOV.B R0,@-R1
MOV.W 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.	Rn - 2 → Rn, Rm → (Rn)	0010nnnnmmmm0101	-	1	1	MOV.W R0,@-R1
MOV.L 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.	Rn - 4 → Rn, Rm → (Rn)	0010nnnnmmmm0110	-	1	1	MOV.L R0,@-R1
MOV.B @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) → sign ext → Rn, Rm + 1 → Rm	0110nnnnmmmm0100	-	1	1	MOV.B @R0,R1
MOV.W @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) → sign ext → Rn, Rm + 2 → Rm	0110nnnnmmmm0101	-	1	1	MOV.W @R0,R1
MOV.L @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, Rm + 4 → Rm	0110nnnnmmmm0110	-	1	1	MOV.L @R0,R1
MOV.B Rm,@(R0,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 → (R0 + Rn)	0000nnnnmmmm0100	-	1	1	MOV.B R1,@(R0,R2)
MOV.W Rm,@(R0,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 → (R0 + Rn)	0000nnnnmmmm0101	-	1	1	MOV.W R1,@(R0,R2)
MOV.L Rm,@(R0,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 → (R0 + Rn)	0000nnnnmmmm0110	-	1	1	MOV.L R1,@(R0,R2)
MOV.B @(R0,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.	(R0 + Rm) → sign extension → Rn	0000nnnnmmmm1100	-	1	1	MOV.B @(R0,R2),R1
MOV.W @(R0,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.	(R0 + Rm) → sign extension → Rn	0000nnnnmmmm1101	-	1	1	MOV.W @(R0,R2),R1
MOV.L @(R0,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.	(R0 + Rm) → Rn	0000nnnnmmmm1110	-	1	1	MOV.L @(R0,R2),R1
MOV #imm,Rn	Move Immediate Data	Stores immediate data, sign-extended to a longword, into general register Rn.	imm → sign extension → Rn	1110nnnnllllllllll	-	1	1	MOV #H80,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	10011nnndddd	-	1	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	1110nnnnddddddddd	-	1	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	11000100ddddddddd	-	1	1	MOV.B @(2,GBR),R0
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	11000101ddddddddd	-	1	1	MOV.W R0,@(1,GBR)
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	11000110ddddddddd	-	1	1	MOV.L R0,@(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)	1100000103ddddddddd	-	1	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)	1100000103ddddddddd	-	1	1	MOV.W R0,@(2,GBR)
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)	1100000103ddddddddd	-	1	1	MOV.L R0,@(3,GBR)
MOV.B R0,@(disp,Rn)	Move Structure Data	Transfers the source operand to the dest. Designed for structure or a stack.	R0 → (disp + Rn)	10000000nnndddd	-	1	1	MOV.B R0,@(HF,R1)
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)	10000001nnndddd	-	1	1	MOV.W R0,@(HF,R1)
MOV.L Rm,@(disp,Rn)	Move Structure Data	Transfers the source operand to the dest. Designed for structure or a stack.	Rn → (disp × 4 + Rn)	00011nnnnmmmmddd	-	1	1	MOV.L R0,@(HF,R1)
MOV.B @Rm,@(disp,Rn)	Move Structure Data	Transfers the source operand to the dest. Designed for structure or a stack.	(disp + Rn) → sign extension → R0	10000100mmmmddd	-	1	1	MOV.B @R0,@(HF,R1)
MOV.W @Rm,@(disp,Rn)	Move Structure Data	Transfers the source operand to the dest. Designed for structure or a stack.	(disp × 2 + Rn) → sign extension → R0	10000101mmmmddd	-	1	1	MOV.W @R0,@(HF,R1)
MOV.L @Rm,@(disp,Rn)	Move Structure Data	Transfers the source operand to the dest. Designed for structure or a stack.	(disp × 4 + Rn) → Rn	01011nnnnmmmmddd	-	1	1	MOV.L @R0,@(HF,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	11000111ddddddddd	-	1	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	1	MOVT R0
MUL.L Rm,Rn	Multiply Long	Performs 32-bit multiplication of Rn and Rm (Signed or unsigned), and stores the bottom 32 bits of the result in the MACL register. MACH is unchanged	Rn × Rm → MACL	0000nnnnmmmm0111	-	2-4	1	MUL.L R0,R1
MULS.W Rm,Rn	Multiply as Signed Word	Performs 16-bit multiplication of signed Rn and Rm, and stores the 32-bit result in MACL. MACH is unchanged	Signed operation, Rn × Rm → MACL	0010nnnnmmmm1111	-	1-3	1	MULS.W R0,R1
MULU.W Rm,Rn	Multiply							

<b>RTS</b>	Return from Subroutine	Returns from a subroutine procedure. The PC value is restored from PR. 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	Y	-	2	RTS
<b>SETT</b>	Set T Bit	Sets the T bit to 1.	1 → T	0000000000010000	1	1		SETT
<b>SHAL Rn</b>	Shift Arithmetic Left 1 Bit with carry	Arithmetically 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	<b>MSB</b>	1		SHAL R0
<b>SHAR Rn</b>	Shift Arithmetic Right 1 Bit with carry	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	<b>LSB</b>	1		SHAR R0
<b>SHLL Rn</b>	Shift Logical Left 1 Bit with carry	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	<b>MSB</b>	1		SHLL R0
<b>SHLL2 Rn</b>	Shift Logical Left 2 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
<b>SHLL8 Rn</b>	Shift Logical Left 8 Bits		Rn << 8 → Rn	0100nnnn00011000	-	1		SHLL8 R0
<b>SHLL16 Rn</b>	Shift Logical Left 16 Bits		Rn << 16 → Rn	0100nnnn00101000	-	1		SHLL16 R0
<b>SHLR Rn</b>	Shift Logical Right 1 Bit with carry	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	<b>LSB</b>	1		SHLR R0
<b>SHLR2 Rn</b>	Shift Logical Right 2 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
<b>SHLR8 Rn</b>	Shift Logical Right 8 Bits		Rn >> 8 → Rn	0100nnnn00011001	-	1		SHLR8 R0
<b>SHLR16 Rn</b>	Shift Logical Right 16 Bits		Rn >> 16 → Rn	0100nnnn00101001	-	1		SHLR16 R0
<b>SLEEP</b>	Sleep	Sets the CPU into power-down mode. CPU waits for an interrupt request.	Sleep	0000000000010111	-	3		SLEEP
<b>STC SR,Rn</b>	Store Control Register	Stores control register into a specified destination.	SR → Rn	0000nnnn00000010	-	1		STC SR,R0
<b>STC GBR,Rn</b>	Store Control Register	Stores control register into a specified destination.	GBR → Rn	0000nnnn00010010	-	1		
<b>STC VBR,Rn</b>	Store Control Register	Stores control register into a specified destination.	VBR → Rn	0000nnnn00100010	-	1		
<b>STC.L SR,@Rn</b>	Store Control Register	Stores control register into a specified destination.	Rn ← 4 → Rn, SR → (Rn)	0100nnnn00000011	-	2		
<b>STC.L GBR,@Rn</b>	Store Control Register	Stores control register into a specified destination.	Rn ← 4 → Rn, GBR → (Rn)	0100nnnn00010011	-	2		STC.L GBR,@R15
<b>STC.L VBR,@Rn</b>	Store Control Register	Stores control register into a specified destination.	Rn ← 4 → Rn, VBR → (Rn)	0100nnnn00100011	-	2		
<b>STS MACH,Rn</b>	Store System Register	Stores data from system register into a specified destination.	MACH → Rn	0000nnnn00001010	-	1		STS MACH,R0
<b>STS MACL,Rn</b>	Store System Register	Stores data from system register into a specified destination.	MACL → Rn	0000nnnn00011010	-	1		
<b>STS PR,Rn</b>	Store System Register	Stores data from system register into a specified destination.	PR → Rn	0000nnnn00101010	-	1		
<b>STS.L MACH,@-Rn</b>	Store System Register	Stores data from system register into a specified destination.	Rn ← 4 → Rn, MACH → (Rn)	0100nnnn00000010	-	1		
<b>STS.L MACL,@-Rn</b>	Store System Register	Stores data from system register into a specified destination.	Rn ← 4 → Rn, MACL → (Rn)	0100nnnn00010010	-	1		
<b>STS.L PR,@-Rn</b>	Store System Register	Stores data from system register into a specified destination.	Rn ← 4 → Rn, PR → (Rn)	0100nnnn00100010	-	1		STS.L PR,@R15
<b>SUB Rm,Rn</b>	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
<b>SUBC Rm,Rn</b>	Subtract with Carry	Subtracts Rm data and the T bit value from Rn data, and stores the result in Rn. The T bit changes according to the result.	Rn - Rm - T → Rn, Borrow → T	0011nnnnmmmm1010	-	1		SUBC R3,R1
<b>SUBV Rm,Rn</b>	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			<b>Under 1 Flow</b>	SUBV R0,R1
<b>SWAP.B Rm,Rn</b>	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 Rm → Swap upper and lower word → Rn	0110nnnnmmmm1000 0110nnnnmmmm1001	-	1		SWAP.B R0,R1 SWAP.W R0,R1
<b>TAS.B @Rn</b>	Test and Set	Reads byte data from address Rn, and sets the T bit to 1 if the data is 0, or clears the T bit to 0 otherwise. Data bit 7 of (Rn) is then set to 1 (Whatever happened to T). During this operation, the bus is not released.	When (Rn) is 0, 1 → T, 1 → MSB of (Rn)	0100nnnn00011011	<b>reslt</b>	4		TAS.B @R7
<b>TRAPA #imm</b>	Trap Always	Execute Trap imm from vector table specified by VBR (offset=imm*4) The PC and SR values are stored on the stack	PC:SR → Stack area, (imm × 4 + VBR) → PC	11000011iiiiiiii	-	8		TRAPA #H20
<b>TST Rm,Rn</b>	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 R0 & imm, when result is 0, 1 → T	0010nnnnmmmm1000 1100100011iiiiiiii	<b>reslt</b>	1		TST R0,R0 TST #H0,R0
<b>TST #imm, @R0,GBR</b>	Test Logical		(R0 + GBR) & imm, when result is 0, 1 → T	1100110011iiiiiiii	<b>reslt</b>	1		TST.B #H45,@R0,GBR
<b>XOR Rm,Rn</b>	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
<b>XOR #imm,R0</b>	Exclusive OR Logical		R0 ^ imm → R0	1100101011iiiiiiii	-	1		XOR #HF0,R0
<b>XOR.B #imm,@R0,GBR</b>	Exclusive OR Logical		(R0 + GBR) ^ imm → (R0 + GBR)	1100111011iiiiiiii	-	3		XOR.B #H45,@R0,GBR
<b>XTRCT Rm,Rn</b>	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

## SH-DSP only

Opcode	Instruction	Description	Function	Code	Delay Slot	Tbit	Cycl	Example
<b>LDC Rm,MOD</b>	Load to Control Register	Store the source operand into control register	Rm → MOD	0100mmmm01011110	-	1		
<b>LDC Rm,RE</b>	Load to Control Register	Store the source operand into control register	Rm → RE	0100mmmm01111110	-	1		
<b>LDC Rm,RS</b>	Load to Control Register	Store the source operand into control register	Rm → RS	0100mmmm01101110	-	1		
<b>LDC.L @Rm+,MOD</b>	Load to Control Register	Store the source operand into control register	(Rm) → MOD, Rm + 4 → Rm	0100mmmm01010111	-	3		
<b>LDC.L @Rm+,RE</b>	Load to Control Register	Store the source operand into control register	(Rm) → RE, Rm + 4 → Rm	0100mmmm01110111	-	3		
<b>LDC.L @Rm+,RS</b>	Load to Control Register	Store the source operand into control register	(Rm) → RS, Rm + 4 → Rm	0100mmmm01100111	-	3		
<b>LDRE @(disp,PC)</b>	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. use with SETRC	disp × 2 + PC → RE	10001110ddddddd	-	1		LDRE END
<b>LDRS @(disp,PC)</b>	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. use with SETRC	disp × 2 + PC → RS	10001100ddddddd	-	1		LDRS STA
<b>LDS Rm,PR</b>	Load to System Register		Rm → PR	0100mmmm00101010	-	1		LDS R0,PR
<b>LDS Rm,DSR</b>	Load to System Register		Rm → DSR	0100mmmm01101010	-	1		
<b>LDS Rm,A0</b>	Load to System Register		Rm → A0	0100mmmm01111010	-	1		
<b>LDS Rm,X0</b>	Load to System Register		Rm → X0	0100mmmm10001010	-	1		
<b>LDS Rm,X1</b>	Load to System Register		Rm → X1	0100mmmm10011010	-	1		
<b>LDS Rm,Y0</b>	Load to System Register		Rm → Y0	0100mmmm10101010	-	1		
<b>LDS Rm,Y1</b>	Load to System Register		Rm → Y1	0100mmmm10111010	-	1		
<b>LDS.L @Rm+,DSR</b>	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) → DSR, Rm + 4 → Rm	0100mmmm01100110	-	1		
<b>LDS.L @Rm+,A0</b>	Load to System Register		(Rm) → A0, Rm + 4 → Rm	0100mmmm01110110	-	1		
<b>LDS.L @Rm+,X0</b>	Load to System Register		(Rm) → X0, Rm + 4 → Rm	0100nnnn10000110	-	1		
<b>LDS.L @Rm+,X1</b>	Load to System Register		(Rm) → X1, Rm + 4 → Rm	0100nnnn10010110	-	1		
<b>LDS.L @Rm+,Y0</b>	Load to System Register		(Rm) → Y0, Rm + 4 → Rm	0100nnnn10100110	-	1		
<b>LDS.L @Rm+,Y1</b>	Load to System Register		(Rm) → Y1, Rm + 4 → Rm	0100nnnn10110110	-	1		
<b>SETRC Rm</b>	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, RFO 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, RFO imm → RC [23:26] zero → SR[27:24]. Repeat control flag → RF1, RFO	0100mmmm00010100 10000010iiiiiiii	-	1		SETRC #32
<b>STC RE,Rn</b>	Store Control Register	Stores control register into a specified destination.	RE → Rn	0000nnnn01110010	-	1		
<b>STC RS,Rn</b>	Store Control Register	Stores control register into a specified destination.	RS → Rn	0000nnnn01100010	-	1		
<b>STC.L MOD,@Rn</b>	Store Control Register	Stores control register into a specified destination.	Rn ← 4 → Rn, MOD → (Rn)	0100nnnn01000111	-	2		
<b>STC.L RE,@Rn</b>	Store Control Register	Stores control register into a specified destination.	Rn ← 4 → Rn, RE → (Rn)	0100nnnn01110011	-	2		
<b>STC.L RS,@Rn</b>	Store Control Register	Stores control register into a specified destination.	Rn ← 4 → Rn, RS → (Rn)	0100nnnn01100011	-	2		
<b>STS DSR,Rn</b>	Store System Register	Stores data from system register into a specified destination	DSR → Rn	0000nnnn01101010	-	1		
<b>STS A0,Rn</b>	Store System Register	Stores data from system register into a specified destination	A0 → Rn	0000nnnn01111010	-	1		
<b>STS X0,Rn</b>	Store System Register	Stores data from system register into a specified destination	X0 → Rn	0000nnnn10001010	-	1		
<b>STS X1,Rn</b>	Store System Register	Stores data from system register into a specified destination	X1 → Rn	0000nnnn10011010	-	1		
<b>STS Y0,Rn</b>	Store System Register	Stores data from system register into a specified destination	Y0 → Rn	0000nnnn10101010	-	1		
<b>STS Y1,Rn</b>	Store System Register	Stores data from system register into a specified destination	Y1 → Rn	0000nnnn10111010	-	1		
<b>STS.L DSR,@Rn</b>	Store System Register	Stores data from system register into a specified destination	Rn ← 4 → Rn, DSR → (Rn)	0100nnnn01100010	-	1		
<b>STS.L A0,@Rn</b>	Store System Register	Stores data from system register into a specified destination	Rn ← 4 → Rn, A0 → (Rn)	0100nnnn01110010	-	1		
<b>STS.L X0,@Rn</b>	Store System Register	Stores data from system register into a specified destination	Rn ← 4 → Rn, X0 → (Rn)	0100nnnn10000010	-	1		
<b>STS.L X1,@Rn</b>	Store System Register	Stores data from system register into a specified destination	Rn ← 4 → Rn, X1 → (Rn)	0100nnnn10010010	-	1		
<b>STS.L Y0,@Rn</b>	Store System Register	Stores data from system register into a specified destination	Rn ← 4 → Rn, Y0 → (Rn)	0100nnnn10100010	-	1		
<b>STS.L Y1,@Rn</b>	Store System Register	Stores data from system register into a specified destination	Rn ← 4 → Rn, Y1 → (Rn)	0100nnnn10110010	-	1		

# Bases 1

Ctrl	Dec	Neg	Hex	Oct	Binary	Asc	Ebd	Dec	Neg	Hex	Oct	Binary	Asc	Ebd	Dec	Neg	Hex	Oct	Binary	Asc	Ebd	Dec	Neg	Hex	Oct	Binary	Asc	Ebd	
NULL	0	0	00	0	00000000	NUL		32	-224	20	040	00100000	DS		64	-192	40	100	01000000	SP		96	###	60	140	01100000	`	-	
SOH	1	-255	01	1	00000001	SOH	␣	33	-223	21	041	00100001	SOS	!	65	-191	41	101	01000001	A	B		97	###	61	141	01100001	a	/
STX	2	-254	02	2	00000010	STX	␣	34	-222	22	042	00100010	FS	"	66	-190	42	102	01000010	B	C		98	###	62	142	01100010	b	c
ETX	3	-253	03	3	00000011	ETX	␣	35	-221	23	043	00100011	WUS	#	67	-189	43	103	01000011	C	D		99	###	63	143	01100011	B	c
EOT	4	-252	04	4	00000100	SEL	␣	36	-220	24	044	00100100	BYP	\$	68	-188	44	104	01000100	D	E		100	###	64	144	01100100	d	e
ENQ	5	-251	05	5	00000101	HT	␣	37	-219	25	045	00100101	LF	%	69	-187	45	105	01000101	E	F		101	###	65	145	01100101	e	f
ACK	6	-250	06	6	00000110	RNL	␣	38	-218	26	046	00100110	ETB	&	70	-186	46	106	01000110	F	G		102	###	66	146	01100110	f	g
BEL	7	-249	07	7	00000111	DEL	␣	39	-217	27	047	00100111	ESC	'	71	-185	47	107	01000111	G	H		103	###	67	147	01100111	g	h
BS	8	-248	08	10	00001000	GE	␣	40	-216	28	050	00101000	SA	(	72	-184	48	110	01001000	H	I		104	###	68	150	01101000	h	i
TAB	9	-247	09	11	00001001	SPS	␣	41	-215	29	051	00101001	SFE	)	73	-183	49	111	01001001	I	J		105	###	69	151	01101001	i	j
LF	10	-246	0A	12	00001010	RPT	␣	42	-214	2A	052	00101010	SM	*	74	-182	4A	112	01001010	J	K		106	###	6A	152	01101010	j	k
VT	11	-245	0B	13	00001011	VT	␣	43	-213	2B	053	00101011	CSP	+	75	-181	4B	113	01001011	K	L		107	###	6B	153	01101011	k	l
FF	12	-244	0C	14	00001100	FF	␣	44	-212	2C	054	00101100	MFA	,	76	-180	4C	114	01001100	L	M		108	###	6C	154	01101100	l	m
CR	13	-243	0D	15	00001101	CR	␣	45	-211	2D	055	00101101	ENQ	-	77	-179	4D	115	01001101	M	N		109	###	6D	155	01101101	m	n
SO	14	-242	0E	16	00001110	SOH	␣	46	-210	2E	056	00101110	ACK	.	78	-178	4E	116	01001110	N	O		110	###	6E	156	01101110	n	o
SI	15	-241	0F	17	00001111	SI	␣	47	-209	2F	057	00101111	BEL	/	79	-177	4F	117	01001111	O	P		111	###	6F	157	01101111	o	p
DLE	16	-240	10	20	00010000	DLE	␣	48	-208	30	060	00110000		0	80	-176	50	120	01010000	P	Q		112	###	70	160	01110000	p	q
DC1	17	-239	11	21	00010001	DC1	␣	49	-207	31	061	00110001	1	1	81	-175	51	121	01010001	Q	R		113	###	71	161	01110001	q	r
DC2	18	-238	12	22	00010010	DC2	␣	50	-206	32	062	00110010	2	2	82	-174	52	122	01010010	R	S		114	###	72	162	01110010	r	s
DC3	19	-237	13	23	00010011	DC3	␣	51	-205	33	063	00110011	3	3	83	-173	53	123	01010011	S	T		115	###	73	163	01110011	s	t
DC4	20	-236	14	24	00010100	RES	␣	52	-204	34	064	00110100	4	4	84	-172	54	124	01010100	T	U		116	###	74	164	01110100	t	u
NAK	21	-235	15	25	00010101	NL	␣	53	-203	35	065	00110101	5	5	85	-171	55	125	01010101	U	V		117	###	75	165	01110101	u	v
SYN	22	-234	16	26	00010110	BS	␣	54	-202	36	066	00110110	6	6	86	-170	56	126	01010110	V	W		118	###	76	166	01110110	v	w
ETB	23	-233	17	27	00010111	POC	␣	55	-201	37	067	00110111	7	7	87	-169	57	127	01010111	W	X		119	###	77	167	01110111	w	x
CAN	24	-232	18	30	00011000	CAN	␣	56	-200	38	070	00111000	8	8	88	-168	58	130	01011000	X	Y		120	###	78	170	01111000	x	y
EN	25	-231	19	31	00011001	EM	␣	57	-199	39	071	00111001	9	9	89	-167	59	131	01011001	Y	Z		121	###	79	171	01111001	y	z
SUB	26	-230	1A	32	00011010	UBS	␣	58	-198	3A	072	00111010			90	-166	5A	132	01011010	Z	[		122	###	7A	172	01111010	z	{
ESC	27	-229	1B	33	00011011	CU1	␣	59	-197	3B	073	00111011	;	;	91	-165	5B	133	01011011	[	\		123	###	7B	173	01111011	{	}
DS	28	-228	1C	34	00011100	IFS	␣	60	-196	3C	074	00111100	<	<	92	-164	5C	134	01011100	\	]		124	###	7C	174	01111100	}	~
GS	29	-227	1D	35	00011101	IGS	␣	61	-195	3D	075	00111101	=	=	93	-163	5D	135	01011101	]	^		125	###	7D	175	01111101	~	'
RS	30	-226	1E	36	00011110	IRS	␣	62	-194	3E	076	00111110	?	?	94	-162	5E	136	01011110	^	_		126	###	7E	176	01111110	'	"
US	31	-225	1F	37	00011111	IUS	␣	63	-193	3F	077	00111111	?	?	95	-161	5F	137	01011111	_	~		127	###	7F	177	01111111	"	Δ

Dec	Neg	Hex	Oct	Binary	Asc	Ebd	Dec	Neg	Hex	Oct	Binary	Asc	Ebd	Dec	Neg	Hex	Oct	Binary	Asc	Ebd	Dec	Neg	Hex	Oct	Binary	Asc	Ebd		
128	-128	80	200	10000000	␣		160	-96	A0	240	10100000	ā	ā	192	-64	C0	300	11000000	L	l		224	-32	E0	340	11100000	α	\	
129	-127	81	201	10000001	ü	a	161	-95	A1	241	10100001	ī	~	193	-63	C1	301	11000001	l	A	B		225	-31	E1	341	11100001	β	π
130	-126	82	202	10000010	ē	b	162	-94	A2	242	10100010	ō	s	194	-62	C2	302	11000010	t	T		226	-30	E2	342	11100010	Γ	Σ	
131	-125	83	203	10000011	ā	c	163	-93	A3	243	10100011	ū	t	195	-61	C3	303	11000011	t	B	C		227	-29	E3	343	11100011	π	Τ
132	-124	84	204	10000100	ä	d	164	-92	A4	244	10100100	ñ	u	196	-60	C4	304	11000100	-	D	E		228	-28	E4	344	11100100	Σ	U
133	-123	85	205	10000101	ā	e	165	-91	A5	245	10100101	ñ	v	197	-59	C5	305	11000101	-	F	G		229	-27	E5	345	11100101	σ	V
134	-122	86	206	10000110	ä	f	166	-90	A6	246	10100110	ē	w	198	-58	C6	306	11000110	-	H	H		230	-26	E6	346	11100110	ϕ	W
135	-121	87	207	10000111	ä	g	167	-89	A7	247	10100111	ē	x	199	-57	C7	307	11000111	-	I	I		231	-25	E7	347	11100111	ϕ	X
136	-120	88	210	10001000	ē	h	168	-88	A8	250	10101000	ē	y	200	-56	C8	310	11001000	-	I	I		232	-24	E8	350	11101000	ϕ	Y
137	-119	89	211	10001001	ē	i	169	-87	A9	251	10101001	ē	z	201	-55	C9	311	11001001	-	I	I		233	-23	E9	351	11101001	ϕ	Z
138	-118	8A	212	10001010	ē		170	-86	AA	252	10101010			202	-54	CA	312	11001010	-	I	I		234	-22	EA	352	11101010	ϕ	
139	-117	8B	213	10001011	ē		171	-85	AB	253	10101011	½		203	-53	CB	313	11001011	-	I	I		235	-21	EB	353	11101011	ϕ	
140	-116	8C	214	10001100	ē		172	-84	AC	254	10101100	¼		204	-52	CC	314	11001100	-	I	I		236	-20	EC	354	11101100	ϕ	
141	-115	8D	215	10001101	ē		173	-83	AD	255	10101101	¼		205	-51	CD	315	11001101	-	I	I		237	-19	ED	355	11101101	ϕ	
142	-114	8E	216	10001110	ē		174	-82	AE	256	10101110	¼		206	-50	CE	316	11001110	-	I	I		238	-18	EE	356	11101110	ϕ	
143	-113	8F	217	10001111	ē	±	175	-81	AF	257	10101111	¼		207	-49	CF	317	11001111	-	I	I		239	-17	EF	357	11101111	ϕ	
144	-112	90	220	10010000	ē		176	-80	B0	260	10110000	½	^	208	-48	D0	320	11010000	-	I	I		240	-16	F0	360	11110000	ϕ	0
145	-111	91	221	10010001	ē	j	177	-79	B1	261	10110001	½		209	-47	D1	321	11010001	-	I	I		241	-15	F1	361	11110001	ϕ	1
146	-110	92	222	10010010	ē	k	178	-78	B2	262	10110010	½		210	-46	D2	322	11010010	-	I	I		242	-14	F2	362	11110010	ϕ	2
147	-109	93	223	10010011	ē	l	179	-77	B3	263	10110011	½		211	-45	D3	323	11010011	-	I	I		243	-13	F3				

# 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






YUV Formulas
$Y = ((66 * R + 129 * G + 25 * B + 128) >> 8) + 16$ $U = ((-38 * R - 74 * G + 112 * B + 128) >> 8) + 128$ $V = ((112 * R - 94 * G - 18 * B + 128) >> 8) + 128$ $c = Y - 16$ $d = U - 128$ $e = V - 128$ $R = (298 * c + 409 * e + 128) >> 8, \text{ Limit } 0-255$ $G = (298 * c - 100 * d - 208 * e + 128) >> 8, \text{ Limit } 0-255$ $B = (298 * c + 516 * d + 128) >> 8, \text{ Limit } 0-255$

## 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, Computers 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
0	0	0.000	1.000	0.000
15		0.262	0.966	0.268
30	$\pi/6$	0.524	0.866	0.577
45	$\pi/4$	0.785	0.707	1.000
60	$\pi/3$	1.047	0.500	1.732
75		1.309	0.259	3.732
90	$\pi/2$	1.571	0.000	###
105		1.833	-0.259	-3.732
120	$2\pi/3$	2.094	-0.500	-1.732
135	$3\pi/4$	2.356	-0.707	-1.000
150	$5\pi/6$	2.618	-0.866	-0.577
165		2.880	-0.966	-0.268
180	$\pi$	3.142	-1.000	0.000
195		3.403	-0.259	0.268
210		3.665	-0.500	0.577
225		3.927	-0.707	1.000
240		4.189	-0.866	1.732
255		4.451	-0.966	3.732
270	$3\pi/2$	4.712	-1.000	###
285		4.974	-0.966	-3.732
300		5.236	-0.866	-1.732
315		5.498	-0.707	-1.000
330		5.760	-0.500	-0.577
345		6.021	-0.259	-0.268
360	$2\pi$	6.283	0.000	0.000
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
0.000	0.000	1.571	0.000	#NUM!	1.000	#NUM!
0.259	0.262	1.309	0.253	3.864	1.035	3.732
0.500	0.524	1.047	0.464	2.000	1.155	1.732
0.707	0.785	0.785	0.615	1.414	1.414	1.000
0.866	1.047	0.524	0.714	1.155	2.000	0.577
0.966	1.309	0.262	0.768	1.035	3.864	0.268
1.000	1.571	0.000	0.785	1.000	###	0.000
0.966	1.309	0.262	0.768	1.035	-3.864	-0.268
0.866	1.047	0.524	0.714	1.155	-2.000	-0.577
0.707	0.785	0.785	0.615	1.414	-1.414	-1.000
0.500	0.524	1.047	0.464	2.000	-1.155	-1.732
0.259	0.262	1.309	0.253	3.864	-1.035	-3.732
0.000	0.000	1.571	0.000	###	-1.000	###
-0.259	-0.262	1.833	-0.253	-3.864	-1.035	3.732
-0.500	-0.524	2.094	-0.464	-2.000	-1.155	1.732
-0.707	-0.785	2.356	-0.615	-1.414	-1.414	1.000
-0.866	-1.047	2.618	-0.714	-1.155	-2.000	0.577
-0.966	-1.309	2.880	-0.768	-1.035	-3.864	0.268
-1.000	-1.571	3.142	-0.785	-1.000	###	0.000
-0.966	-1.309	2.880	-0.768	-1.035	3.864	-0.268
-0.866	-1.047	2.618	-0.714	-1.155	2.000	-0.577
-0.707	-0.785	2.356	-0.615	-1.414	1.414	-1.000
-0.500	-0.524	2.094	-0.464	-2.000	1.155	-1.732
-0.259	-0.262	1.833	-0.253	-3.864	1.035	-3.732
0.000	0.000	1.571	0.000	###	1.000	###
0.2588	0.2618	1.3090	0.2533			
0.5000	0.5236	1.0472	0.4636			

Byte Rad	Byte Sin
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

$\text{Cos}(X) = \text{Sin}(X + 90^\circ)$

$\text{Csc}(X) = 1/\text{Sin}(X)$

$\text{Cot}(X) = 1/\text{Tan}(X)$

$\text{Sin}(X) = \text{Cos}(X - 90^\circ)$

$\text{Sec}(X) = 1/\text{Cos}(X)$

$\text{Cot}(X) = \text{Cos}(X) / \text{Sin}(X)$

$\text{degrees} = \text{radians} \times 180^\circ / \pi$

$\text{radians} = \text{degrees} \times \pi / 180^\circ$

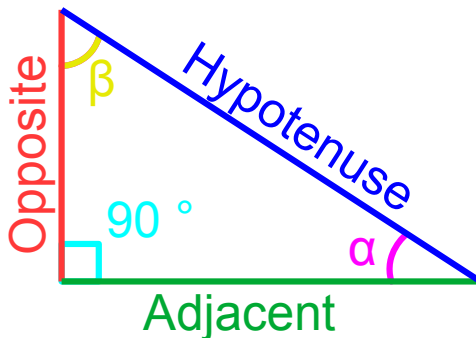
$90 \text{ Degrees} = \pi/2 \text{ rad}$

### Small Angle Approximations

$\text{Sin } a \approx a$

$\text{Cos } a \approx 1 - (a^2) / 2 \approx 1$

$\text{Tan } a \approx a$



Trigonometry  $\text{Sin}(\alpha) = \text{Opposite} / \text{Hypotenuse}$

Trigonometry  $\text{Cos}(\alpha) = \text{Adjacent} / \text{Hypotenuse}$

Trigonometry  $\text{Tan}(\alpha) = \text{Opposite} / \text{Adjacent}$

Trigonometry  $\text{Csc}(\alpha) = \text{Hypotenuse} / \text{Opposite}$

Trigonometry  $\text{Sec}(\alpha) = \text{Hypotenuse} / \text{Adjacent}$

Trigonometry  $\text{Cot}(\alpha) = \text{Adjacent} / \text{Opposite}$

180 Rule  $90 + \alpha + \beta = 180^\circ$

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

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

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

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
Matrix A	K	KA+LF	KB+LG	KC+LH	KD+LI	KE+LJ
	M	MA+NF	MB+NG	MC+NH	MD+NI	ME+NJ
	O	OA+PF	OB+PG	OC+PH	OD+PI	OE+PJ
	Q	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

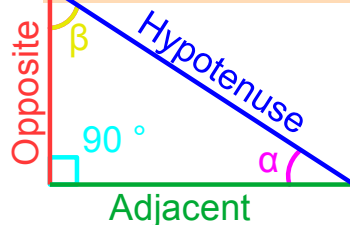
	X	Y	Z
X	cos(A)	0	-sin(A)
Y	0	1	0
Z	sin(A)	0	cos(A)

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 H² = A² + O²

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	√(H² - O²)	√(H² - A²)	√(A² + O²)	
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/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/Rotation_matrix)  
[https://en.wikipedia.org/wiki/Small-angle\\_approximation](https://en.wikipedia.org/wiki/Small-angle_approximation)



Note	C	C # / D ♭	D	D # / E ♭	E	F	F # / G ♭	G	G # / A ♭	A	A # / B ♭	B
Octave 0	16.3516	17.3239	18.3541	19.4454	20.6017	21.8268	23.1247	24.4997	25.9565	27.5000	29.1352	30.8677
Octave 1	32.7032	34.6478	36.7081	38.8909	41.2034	43.6535	46.2493	48.9994	51.9131	55.0000	58.2705	61.7354
Octave 2	65.4064	69.2957	73.4162	77.7818	82.4069	87.3071	92.4986	97.9989	103.8262	110.0000	116.5409	123.4708
Octave 3	130.813	138.591	146.832	155.564	164.814	174.614	184.997	195.998	207.652	220.000	233.082	246.942
Octave 4	261.626	277.183	293.665	311.127	329.628	349.228	369.994	391.995	415.305	440.000	466.164	493.883
Octave 5	523.251	554.365	587.330	622.254	659.255	698.457	739.989	783.991	830.609	880.000	932.328	987.767
Octave 6	1046.50	1108.73	1174.66	1244.51	1318.51	1396.91	1479.98	1567.98	1661.22	1760.00	1864.66	1975.53
Octave 7	2093.01	2217.46	2349.32	2489.02	2637.02	2793.83	2959.96	3135.96	3322.44	3520.00	3729.31	3951.07
Octave 8	4186.01	4434.92	4698.64	4978.03	5274.04	5587.65	5919.91	6271.93	6644.88	7040.00	7458.62	7902.13

Floating Point	16-bit Signed	8bit signed	8 Bit Unsigned	4 Bit Unsigned
1	32767	127	255	15
...	...	...	...	...
0	0	0	128	8
...	...	...	...	...
-1	-32768	-128	0	0

