Alias	- Group -	#S = immediate (I=1). S = register. #D = immediate (L=1). D = register. - Assembly Syntax -	* Z = (result == 0). ** If #S and cogex, PC += signed(S). If #S and hubex, PC += signed(S*4). If S, PC = register S. - Description -
.	Miscellaneous	NOP	No operation.
.	Math and Logic	ROR D,{#}S {WC/WZ/WCZ}	Rotate right. D = [31:0] of ({D[31:0], D[31:0]} >> S[4:0]). C = last bit shifted out if S[4:0] > 0, else D[0]. *
.	Math and Logic	ROL D,{#}S {WC/WZ/WCZ}	Rotate left. D = [63:32] of ({D[31:0], D[31:0]} << S[4:0]). C = last bit shifted out if S[4:0] > 0, else D[31]. *
.	Math and Logic	SHR D,{#}S {WC/WZ/WCZ}	Shift right. D = [31:0] of ({32'b0, D[31:0]} >> S[4:0]). C = last bit shifted out if S[4:0] > 0, else D[0]. *
.	Math and Logic	SHL D,{#}S {WC/WZ/WCZ}	Shift left. D = [63:32] of ({D[31:0], 32'b0} << S[4:0]). C = last bit shifted out if S[4:0] > 0, else D[31]. *
.	Math and Logic	RCR D,{#}S {WC/WZ/WCZ}	Rotate carry right. D = [31:0] of ({{32{C}}, D[31:0]} >> S[4:0]). C = last bit shifted out if S[4:0] > 0, else D[0]. *
.	Math and Logic	RCL D,{#}S {WC/WZ/WCZ}	Rotate carry left. D = [63:32] of ({D[31:0], {32{C}}} << S[4:0]). C = last bit shifted out if S[4:0] > 0, else D[31]. *
.	Math and Logic	SAR D,{#}S {WC/WZ/WCZ}	Shift arithmetic right. D = [31:0] of ({{32{D[31]}}, D[31:0]} >> S[4:0]). C = last bit shifted out if S[4:0] > 0, else D[0]. *
.	Math and Logic	SAL D,{#}S {WC/WZ/WCZ}	Shift arithmetic left. D = [63:32] of ({D[31:0], {32{D[0]}}} << S[4:0]). C = last bit shifted out if S[4:0] > 0, else D[31]. *
.	Math and Logic	ADD D,{#}S {WC/WZ/WCZ}	Add S into D. D = D + S. C = carry of (D + S). *
.	Math and Logic	ADDX D,{#}S {WC/WZ/WCZ}	Add (S + C) into D, extended. D = D + S + C. C = carry of (D + S + C). Z = Z AND (result == 0).
.	Math and Logic	ADDS D,{#}S {WC/WZ/WCZ}	Add S into D, signed. D = D + S. C = correct sign of (D + S). *
.	Math and Logic	ADDSX D,{#}S {WC/WZ/WCZ}	Add (S + C) into D, signed and extended. D = D + S + C. C = correct sign of (D + S + C). Z = Z AND (result == 0).
.	Math and Logic	SUB D,{#}S {WC/WZ/WCZ}	Subtract S from D. D = D - S. C = borrow of (D - S). *
.	Math and Logic	SUBX D,{#}S {WC/WZ/WCZ}	Subtract (S + C) from D, extended. D = D - (S + C). C = borrow of (D - (S + C)). Z = Z AND (result == 0).
.	Math and Logic	SUBS D,{#}S {WC/WZ/WCZ}	Subtract S from D, signed. D = D - S. C = correct sign of (D - S). *
.	Math and Logic	SUBSX D,{#}S {WC/WZ/WCZ}	Subtract (S + C) from D, signed and extended. D = D - (S + C). C = correct sign of (D - (S + C)). Z = Z AND (result == 0).
.	Math and Logic	CMP D,{#}S {WC/WZ/WCZ}	Compare D to S. C = borrow of (D - S). Z = (D == S).
.	Math and Logic	CMPX D,{#}S {WC/WZ/WCZ}	Compare D to (S + C), extended. C = borrow of (D - (S + C)). Z = Z AND (D == S + C).
.	Math and Logic	CMPS D,{#}S {WC/WZ/WCZ}	Compare D to S, signed. C = correct sign of (D - S). Z = (D == S).
.	Math and Logic	CMPSX D,{#}S {WC/WZ/WCZ}	Compare D to (S + C), signed and extended. C = correct sign of (D - (S + C)). Z = Z AND (D == S + C).
.	Math and Logic	CMPR D,{#}S {WC/WZ/WCZ}	Compare S to D (reverse). C = borrow of (S - D). Z = (D == S).
.	Math and Logic	CMPM D,{#}S {WC/WZ/WCZ}	Compare D to S, get MSB of difference into C. C = MSB of (D - S). Z = (D == S).
.	Math and Logic	SUBR D,{#}S {WC/WZ/WCZ}	Subtract D from S (reverse). D = S - D. C = borrow of (S - D). *
.	Math and Logic	CMPSUB D,{#}S {WC/WZ/WCZ}	Compare and subtract S from D if D >= S. If D => S then D = D - S and C = 1, else D same and C = 0. *
.	Math and Logic	FGE D,{#}S {WC/WZ/WCZ}	Force D >= S. If D < S then D = S and C = 1, else D same and C = 0. *
.	Math and Logic	FLE D,{#}S {WC/WZ/WCZ}	Force D <= S. If D > S then D = S and C = 1, else D same and C = 0. *
.	Math and Logic	FGES D,{#}S {WC/WZ/WCZ}	Force D >= S, signed. If D < S then D = S and C = 1, else D same and C = 0. *
.	Math and Logic	FLES D,{#}S {WC/WZ/WCZ}	Force D <= S, signed. If D > S then D = S and C = 1, else D same and C = 0. *
.	Math and Logic	SUMC D,{#}S {WC/WZ/WCZ}	Sum +/-S into D by C. If C = 1 then D = D - S, else D = D + S. C = correct sign of (D +/- S). *
.	Math and Logic	SUMNC D,{#}S {WC/WZ/WCZ}	Sum +/-S into D by !C. If C = 0 then D = D - S, else D = D + S. C = correct sign of (D +/- S). *
.	Math and Logic	SUMZ D,{#}S {WC/WZ/WCZ}	Sum +/-S into D by Z. If Z = 1 then D = D - S, else D = D + S. C = correct sign of (D +/- S). *
.	Math and Logic	SUMNZ D,{#}S {WC/WZ/WCZ}	Sum +/-S into D by !Z. If Z = 0 then D = D - S, else D = D + S. C = correct sign of (D +/- S). *
.	Math and Logic	TESTB D,{#}S WC/WZ	Test bit S[4:0] of D, write to C/Z. C/Z = D[S[4:0]].
.	Math and Logic	TESTBN D,{#}S WC/WZ	Test bit S[4:0] of !D, write to C/Z. C/Z = !D[S[4:0]].
.	Math and Logic	TESTB D,{#}S ANDC/ANDZ	Test bit S[4:0] of D, AND into C/Z. C/Z = C/Z AND D[S[4:0]].
.	Math and Logic	TESTBN D,{#}S ANDC/ANDZ	Test bit S[4:0] of !D, AND into C/Z. C/Z = C/Z AND !D[S[4:0]].
.	Math and Logic	TESTB D,{#}S ORC/ORZ	Test bit S[4:0] of D, OR into C/Z. C/Z = C/Z OR D[S[4:0]].
.	Math and Logic	TESTBN D,{#}S ORC/ORZ	Test bit S[4:0] of !D, OR into C/Z. C/Z = C/Z OR !D[S[4:0]].
.	Math and Logic	TESTB D,{#}S XORC/XORZ	Test bit S[4:0] of D, XOR into C/Z. C/Z = C/Z XOR D[S[4:0]].
.	Math and Logic	TESTBN D,{#}S XORC/XORZ	Test bit S[4:0] of !D, XOR into C/Z. C/Z = C/Z XOR !D[S[4:0]].
.	Math and Logic	BITL D,{#}S {WCZ}	Bits D[S[9:5]+S[4:0]:S[4:0]] = 0. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]].
.	Math and Logic	BITH D,{#}S {WCZ}	Bits D[S[9:5]+S[4:0]:S[4:0]] = 1. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]].

.	Math and Logic	BITC	D,{#}S	{WCZ}	Bits D[S[9:5]+S[4:0]:S[4:0]] = C. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]].
.	Math and Logic	BITNC	D,{#}S	{WCZ}	Bits D[S[9:5]+S[4:0]:S[4:0]] = !C. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]].
.	Math and Logic	BITZ	D,{#}S	{WCZ}	Bits D[S[9:5]+S[4:0]:S[4:0]] = Z. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]].
.	Math and Logic	BITNZ	D,{#}S	{WCZ}	Bits D[S[9:5]+S[4:0]:S[4:0]] = !Z. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]].
.	Math and Logic	BITRND	D,{#}S	{WCZ}	Bits D[S[9:5]+S[4:0]:S[4:0]] = RNDs. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]].
.	Math and Logic	BITNOT	D,{#}S	{WCZ}	Toggle bits D[S[9:5]+S[4:0]:S[4:0]]. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]].
.	Math and Logic	AND	D,{#}S	{WC/WZ/WCZ}	AND S into D. D = D & S. C = parity of result. *
.	Math and Logic	ANDN	D,{#}S	{WC/WZ/WCZ}	AND !S into D. D = D & !S. C = parity of result. *
.	Math and Logic	OR	D,{#}S	{WC/WZ/WCZ}	OR S into D. D = D | S. C = parity of result. *
.	Math and Logic	XOR	D,{#}S	{WC/WZ/WCZ}	XOR S into D. D = D ^ S. C = parity of result. *
.	Math and Logic	MUXC	D,{#}S	{WC/WZ/WCZ}	Mux C into each D bit that is '1' in S. D = (!S & D) | (S & {32{ C}}). C = parity of result. *
.	Math and Logic	MUXNC	D,{#}S	{WC/WZ/WCZ}	Mux !C into each D bit that is '1' in S. D = (!S & D) | (S & {32{!C}}). C = parity of result. *
.	Math and Logic	MUXZ	D,{#}S	{WC/WZ/WCZ}	Mux Z into each D bit that is '1' in S. D = (!S & D) | (S & {32{ Z}}). C = parity of result. *
.	Math and Logic	MUXNZ	D,{#}S	{WC/WZ/WCZ}	Mux !Z into each D bit that is '1' in S. D = (!S & D) | (S & {32{!Z}}). C = parity of result. *
.	Math and Logic	MOV	D,{#}S	{WC/WZ/WCZ}	Move S into D. D = S. C = S[31]. *
.	Math and Logic	NOT	D,{#}S	{WC/WZ/WCZ}	Get !S into D. D = !S. C = !S[31]. *
alias	Math and Logic	NOT	D	{WC/WZ/WCZ}	Get !D into D. D = !D. C = !D[31]. *
.	Math and Logic	ABS	D,{#}S	{WC/WZ/WCZ}	Get absolute value of S into D. D = ABS(S). C = S[31]. *
alias	Math and Logic	ABS	D	{WC/WZ/WCZ}	Get absolute value of D into D. D = ABS(D). C = D[31]. *
.	Math and Logic	NEG	D,{#}S	{WC/WZ/WCZ}	Negate S into D. D = -S. C = MSB of result. *
alias	Math and Logic	NEG	D	{WC/WZ/WCZ}	Negate D. D = -D. C = MSB of result. *
.	Math and Logic	NEGC	D,{#}S	{WC/WZ/WCZ}	Negate S by C into D. If C = 1 then D = -S, else D = S. C = MSB of result. *
alias	Math and Logic	NEGC	D	{WC/WZ/WCZ}	Negate D by C. If C = 1 then D = -D, else D = D. C = MSB of result. *
.	Math and Logic	NEGNC	D,{#}S	{WC/WZ/WCZ}	Negate S by !C into D. If C = 0 then D = -S, else D = S. C = MSB of result. *
alias	Math and Logic	NEGNC	D	{WC/WZ/WCZ}	Negate D by !C. If C = 0 then D = -D, else D = D. C = MSB of result. *
.	Math and Logic	NEGZ	D,{#}S	{WC/WZ/WCZ}	Negate S by Z into D. If Z = 1 then D = -S, else D = S. C = MSB of result. *
alias	Math and Logic	NEGZ	D	{WC/WZ/WCZ}	Negate D by Z. If Z = 1 then D = -D, else D = D. C = MSB of result. *
.	Math and Logic	NEGNZ	D,{#}S	{WC/WZ/WCZ}	Negate S by !Z into D. If Z = 0 then D = -S, else D = S. C = MSB of result. *
alias	Math and Logic	NEGNZ	D	{WC/WZ/WCZ}	Negate D by !Z. If Z = 0 then D = -D, else D = D. C = MSB of result. *
.	Math and Logic	INCMOD	D,{#}S	{WC/WZ/WCZ}	Increment with modulus. If D = S then D = 0 and C = 1, else D = D + 1 and C = 0. *
.	Math and Logic	DECMOD	D,{#}S	{WC/WZ/WCZ}	Decrement with modulus. If D = 0 then D = S and C = 1, else D = D - 1 and C = 0. *
.	Math and Logic	ZEROX	D,{#}S	{WC/WZ/WCZ}	Zero-extend D above bit S[4:0]. C = MSB of result. *
.	Math and Logic	SIGNX	D,{#}S	{WC/WZ/WCZ}	Sign-extend D from bit S[4:0]. C = MSB of result. *
.	Math and Logic	ENCOD	D,{#}S	{WC/WZ/WCZ}	Get bit position of top-most '1' in S into D. D = position of top '1' in S (0..31). C = (S != 0). *
alias	Math and Logic	ENCOD	D	{WC/WZ/WCZ}	Get bit position of top-most '1' in D into D. D = position of top '1' in S (0..31). C = (S != 0). *
.	Math and Logic	ONES	D,{#}S	{WC/WZ/WCZ}	Get number of '1's in S into D. D = number of '1's in S (0..32). C = LSB of result. *
alias	Math and Logic	ONES	D	{WC/WZ/WCZ}	Get number of '1's in D into D. D = number of '1's in S (0..32). C = LSB of result. *
.	Math and Logic	TEST	D,{#}S	{WC/WZ/WCZ}	Test D with S. C = parity of (D & S). Z = ((D & S) == 0).
alias	Math and Logic	TEST	D	{WC/WZ/WCZ}	Test D. C = parity of D. Z = (D == 0).
.	Math and Logic	TESTN	D,{#}S	{WC/WZ/WCZ}	Test D with !S. C = parity of (D & !S). Z = ((D & !S) == 0).
.	Math and Logic	SETNIB	D,{#}S,#N		Set S[3:0] into nibble N in D, keeping rest of D same.
alias	Math and Logic	SETNIB	{#}S		Set S[3:0] into nibble established by prior ALTSN instruction.
.	Math and Logic	GETNIB	D,{#}S,#N		Get nibble N of S into D. D = {28'b0, S.NIBBLE[N]).
alias	Math and Logic	GETNIB	D		Get nibble established by prior ALTGN instruction into D.
.	Math and Logic	ROLNIB	D,{#}S,#N		Rotate-left nibble N of S into D. D = {D[27:0], S.NIBBLE[N]).
alias	Math and Logic	ROLNIB	D		Rotate-left nibble established by prior ALTGN instruction into D.
.	Math and Logic	SETBYTE	D,{#}S,#N		Set S[7:0] into byte N in D, keeping rest of D same.

alias	Math and Logic	SETBYTE {#}S	Set S[7:0] into byte established by prior ALTSB instruction.
.	Math and Logic	GETBYTE D,{#}S,#N	Get byte N of S into D. D = {24'b0, S.BYTE[N]).
alias	Math and Logic	GETBYTE D	Get byte established by prior ALTGB instruction into D.
.	Math and Logic	ROLBYTE D,{#}S,#N	Rotate-left byte N of S into D. D = {D[23:0], S.BYTE[N]).
alias	Math and Logic	ROLBYTE D	Rotate-left byte established by prior ALTGB instruction into D.
.	Math and Logic	SETWORD D,{#}S,#N	Set S[15:0] into word N in D, keeping rest of D same.
alias	Math and Logic	SETWORD {#}S	Set S[15:0] into word established by prior ALTSW instruction.
.	Math and Logic	GETWORD D,{#}S,#N	Get word N of S into D. D = {16'b0, S.WORD[N]).
alias	Math and Logic	GETWORD D	Get word established by prior ALTGW instruction into D.
.	Math and Logic	ROLWORD D,{#}S,#N	Rotate-left word N of S into D. D = {D[15:0], S.WORD[N]).
alias	Math and Logic	ROLWORD D	Rotate-left word established by prior ALTGW instruction into D.
.	Register Indirection	ALTSN D,{#}S	Alter subsequent SETNIB instruction. Next D field = (D[11:3] + S) & $1FF, N field = D[2:0]. D += sign-extended S[17:9].
alias	Register Indirection	ALTSN D	Alter subsequent SETNIB instruction. Next D field = D[11:3], N field = D[2:0].
.	Register Indirection	ALTGN D,{#}S	Alter subsequent GETNIB/ROLNIB instruction. Next S field = (D[11:3] + S) & $1FF, N field = D[2:0]. D += sign-extended S[17:9].
alias	Register Indirection	ALTGN D	Alter subsequent GETNIB/ROLNIB instruction. Next S field = D[11:3], N field = D[2:0].
.	Register Indirection	ALTSB D,{#}S	Alter subsequent SETBYTE instruction. Next D field = (D[10:2] + S) & $1FF, N field = D[1:0]. D += sign-extended S[17:9].
alias	Register Indirection	ALTSB D	Alter subsequent SETBYTE instruction. Next D field = D[10:2], N field = D[1:0].
.	Register Indirection	ALTGB D,{#}S	Alter subsequent GETBYTE/ROLBYTE instruction. Next S field = (D[10:2] + S) & $1FF, N field = D[1:0]. D += sign-extended S[17:9].
alias	Register Indirection	ALTGB D	Alter subsequent GETBYTE/ROLBYTE instruction. Next S field = D[10:2], N field = D[1:0].
.	Register Indirection	ALTSW D,{#}S	Alter subsequent SETWORD instruction. Next D field = (D[9:1] + S) & $1FF, N field = D[0]. D += sign-extended S[17:9].
alias	Register Indirection	ALTSW D	Alter subsequent SETWORD instruction. Next D field = D[9:1], N field = D[0].
.	Register Indirection	ALTGW D,{#}S	Alter subsequent GETWORD/ROLWORD instruction. Next S field = ((D[9:1] + S) & $1FF), N field = D[0]. D += sign-extended S[17:9].
alias	Register Indirection	ALTGW D	Alter subsequent GETWORD/ROLWORD instruction. Next S field = D[9:1], N field = D[0].
.	Register Indirection	ALTR D,{#}S	Alter result register address (normally D field) of next instruction to (D + S) & $1FF. D += sign-extended S[17:9].
alias	Register Indirection	ALTR D	Alter result register address (normally D field) of next instruction to D[8:0].
.	Register Indirection	ALTD D,{#}S	Alter D field of next instruction to (D + S) & $1FF. D += sign-extended S[17:9].
alias	Register Indirection	ALTD D	Alter D field of next instruction to D[8:0].
.	Register Indirection	ALTS D,{#}S	Alter S field of next instruction to (D + S) & $1FF. D += sign-extended S[17:9].
alias	Register Indirection	ALTS D	Alter S field of next instruction to D[8:0].
.	Register Indirection	ALTB D,{#}S	Alter D field of next instruction to (D[13:5] + S) & $1FF. D += sign-extended S[17:9].
alias	Register Indirection	ALTB D	Alter D field of next instruction to D[13:5].
.	Register Indirection	ALTI D,{#}S	Substitute next instruction's I/R/D/S fields with fields from D, per S. Modify D per S.
alias	Register Indirection	ALTI D	Execute D in place of next instruction. D stays same.
.	Math and Logic	SETR D,{#}S	Set R field of D to S[8:0]. D = {D[31:28], S[8:0], D[18:0]}.
.	Math and Logic	SETD D,{#}S	Set D field of D to S[8:0]. D = {D[31:18], S[8:0], D[8:0]}.
.	Math and Logic	SETS D,{#}S	Set S field of D to S[8:0]. D = {D[31:9], S[8:0]}.
.	Math and Logic	DECOD D,{#}S	Decode S[4:0] into D. D = 1 << S[4:0].
alias	Math and Logic	DECOD D	Decode D[4:0] into D. D = 1 << D[4:0].
.	Math and Logic	BMASK D,{#}S	Get LSB-justified bit mask of size (S[4:0] + 1) into D. D = ($0000_0002 << S[4:0]) - 1.
alias	Math and Logic	BMASK D	Get LSB-justified bit mask of size (D[4:0] + 1) into D. D = ($0000_0002 << D[4:0]) - 1.
.	Math and Logic	CRCBIT D,{#}S	Iterate CRC value in D using C and polynomial in S. If (C XOR D[0]) then D = (D >> 1) XOR S, else D = (D >> 1).
.	Math and Logic	CRCNIB D,{#}S	Iterate CRC value in D using Q[31:28] and polynomial in S. Like CRCBIT, but 4x. Q = Q << 4. Use SETQ+CRCNIB+CRCNIB+CRCNIB...
.	Math and Logic	MUXNITS D,{#}S	For each non-zero bit pair in S, copy that bit pair into the corresponding D bits, else leave that D bit pair the same.
.	Math and Logic	MUXNIBS D,{#}S	For each non-zero nibble in S, copy that nibble into the corresponding D nibble, else leave that D nibble the same.
.	Math and Logic	MUXQ D,{#}S	Used after SETQ. For each '1' bit in Q, copy the corresponding bit in S into D. D = (D & !Q) | (S & Q).
.	Math and Logic	MOVBYTS D,{#}S	Move bytes within D, per S. D = {D.BYTE[S[7:6]], D.BYTE[S[5:4]], D.BYTE[S[3:2]], D.BYTE[S[1:0]]}.
.	Math and Logic	MUL D,{#}S {WZ}	D = unsigned (D[15:0] * S[15:0]). Z = (S == 0) | (D == 0).

.	Math and Logic	MULS	D,{#}S	{WZ}	D = signed (D[15:0] * S[15:0]). Z = (S == 0) | (D == 0).
.	Math and Logic	SCA	D,{#}S	{WZ}	Next instruction's S value = unsigned (D[15:0] * S[15:0]) >> 16. *
.	Math and Logic	SCAS	D,{#}S	{WZ}	Next instruction's S value = signed (D[15:0] * S[15:0]) >> 14. In this scheme, $4000 = 1.0 and $C000 = -1.0. *
.	Pixel Mixer	ADDPIX	D,{#}S		Add bytes of S into bytes of D, with $FF saturation.
.	Pixel Mixer	MULPIX	D,{#}S		Multiply bytes of S into bytes of D, where $FF = 1.0.
.	Pixel Mixer	BLNPIX	D,{#}S		Alpha-blend bytes of S into bytes of D, using SETPIV value.
.	Pixel Mixer	MIXPIX	D,{#}S		Mix bytes of S into bytes of D, using SETPIX and SETPIV values.
.	Events - Configuration	ADDCT1	D,{#}S		Set CT1 event to trigger on CT = D + S. Adds S into D.
.	Events - Configuration	ADDCT2	D,{#}S		Set CT2 event to trigger on CT = D + S. Adds S into D.
.	Events - Configuration	ADDCT3	D,{#}S		Set CT3 event to trigger on CT = D + S. Adds S into D.
.	Hub RAM - Write	WMLONG	D,{#}S/P		Write only non-$00 bytes in D[31:0] to hub address {#}S/PTRx. Prior SETQ/SETQ2 invokes cog/LUT block transfer.
.	Smart Pins	RQPIN	D,{#}S	{WC}	Read smart pin S[5:0] result "Z" into D, don't acknowledge smart pin ("Q" in RQPIN means "quiet"). C = modal result.
.	Smart Pins	RDPIN	D,{#}S	{WC}	Read smart pin S[5:0] result "Z" into D, acknowledge smart pin. C = modal result.
.	Lookup Table	RDLUT	D,{#}S/P	{WC/WZ/WCZ}	Read data from LUT address {#}S/PTRx into D. C = MSB of data. *
.	Hub RAM - Read	RDBYTE	D,{#}S/P	{WC/WZ/WCZ}	Read zero-extended byte from hub address {#}S/PTRx into D. C = MSB of byte. *
.	Hub RAM - Read	RDWORD	D,{#}S/P	{WC/WZ/WCZ}	Read zero-extended word from hub address {#}S/PTRx into D. C = MSB of word. *
.	Hub RAM - Read	RDLONG	D,{#}S/P	{WC/WZ/WCZ}	Read long from hub address {#}S/PTRx into D. C = MSB of long. * Prior SETQ/SETQ2 invokes cog/LUT block transfer.
alias	Hub RAM - Read	POPA	D	{WC/WZ/WCZ}	Read long from hub address --PTRA into D. C = MSB of long. *
alias	Hub RAM - Read	POPB	D	{WC/WZ/WCZ}	Read long from hub address --PTRB into D. C = MSB of long. *
.	Branch S - Call	CALLD	D,{#}S	{WC/WZ/WCZ}	Call to S** by writing {C, Z, 10'b0, PC[19:0]} to D. C = S[31], Z = S[30].
alias	Branch S - Resume	RESI3			Resume from INT3. (CALLD $1F0,$1F1 WC,WZ)
alias	Branch S - Resume	RESI2			Resume from INT2. (CALLD $1F2,$1F3 WC,WZ)
alias	Branch S - Resume	RESI1			Resume from INT1. (CALLD $1F4,$1F5 WC,WZ)
alias	Branch S - Resume	RESI0			Resume from INT0. (CALLD $1FE,$1FF WC,WZ)
alias	Branch S - Return	RETI3			Return from INT3. (CALLD $1FF,$1F1 WC,WZ)
alias	Branch S - Return	RETI2			Return from INT2. (CALLD $1FF,$1F3 WC,WZ)
alias	Branch S - Return	RETI1			Return from INT1. (CALLD $1FF,$1F5 WC,WZ)
alias	Branch S - Return	RETI0			Return from INT0. (CALLD $1FF,$1FF WC,WZ)
.	Branch S - Call	CALLPA	{#}D,{#}S		Call to S** by pushing {C, Z, 10'b0, PC[19:0]} onto stack, copy D to PA.
.	Branch S - Call	CALLPB	{#}D,{#}S		Call to S** by pushing {C, Z, 10'b0, PC[19:0]} onto stack, copy D to PB.
.	Branch S - Mod & Test	DJZ	D,{#}S		Decrement D and jump to S** if result is zero.
.	Branch S - Mod & Test	DJNZ	D,{#}S		Decrement D and jump to S** if result is not zero.
.	Branch S - Mod & Test	DJF	D,{#}S		Decrement D and jump to S** if result is $FFFF_FFFF.
.	Branch S - Mod & Test	DJNF	D,{#}S		Decrement D and jump to S** if result is not $FFFF_FFFF.
.	Branch S - Mod & Test	IJZ	D,{#}S		Increment D and jump to S** if result is zero.
.	Branch S - Mod & Test	IJNZ	D,{#}S		Increment D and jump to S** if result is not zero.
.	Branch S - Test	TJZ	D,{#}S		Test D and jump to S** if D is zero.
.	Branch S - Test	TJNZ	D,{#}S		Test D and jump to S** if D is not zero.
.	Branch S - Test	TJF	D,{#}S		Test D and jump to S** if D is full (D = $FFFF_FFFF).
.	Branch S - Test	TJNF	D,{#}S		Test D and jump to S** if D is not full (D != $FFFF_FFFF).
.	Branch S - Test	TJS	D,{#}S		Test D and jump to S** if D is signed (D[31] = 1).
.	Branch S - Test	TJNS	D,{#}S		Test D and jump to S** if D is not signed (D[31] = 0).
.	Branch S - Test	TJV	D,{#}S		Test D and jump to S** if D overflowed (D[31] != C, C = 'correct sign' from last addition/subtraction).
.	Events - Branch	JINT	{#}S		Jump to S** if INT event flag is set.
.	Events - Branch	JCT1	{#}S		Jump to S** if CT1 event flag is set.
.	Events - Branch	JCT2	{#}S		Jump to S** if CT2 event flag is set.
.	Events - Branch	JCT3	{#}S		Jump to S** if CT3 event flag is set.

.	Events - Branch	JSE1	{#}S	Jump to S** if SE1 event flag is set.
.	Events - Branch	JSE2	{#}S	Jump to S** if SE2 event flag is set.
.	Events - Branch	JSE3	{#}S	Jump to S** if SE3 event flag is set.
.	Events - Branch	JSE4	{#}S	Jump to S** if SE4 event flag is set.
.	Events - Branch	JPAT	{#}S	Jump to S** if PAT event flag is set.
.	Events - Branch	JFBW	{#}S	Jump to S** if FBW event flag is set.
.	Events - Branch	JXMT	{#}S	Jump to S** if XMT event flag is set.
.	Events - Branch	JXFI	{#}S	Jump to S** if XFI event flag is set.
.	Events - Branch	JXRO	{#}S	Jump to S** if XRO event flag is set.
.	Events - Branch	JXRL	{#}S	Jump to S** if XRL event flag is set.
.	Events - Branch	JATN	{#}S	Jump to S** if ATN event flag is set.
.	Events - Branch	JQMT	{#}S	Jump to S** if QMT event flag is set.
.	Events - Branch	JNINT	{#}S	Jump to S** if INT event flag is clear.
.	Events - Branch	JNCT1	{#}S	Jump to S** if CT1 event flag is clear.
.	Events - Branch	JNCT2	{#}S	Jump to S** if CT2 event flag is clear.
.	Events - Branch	JNCT3	{#}S	Jump to S** if CT3 event flag is clear.
.	Events - Branch	JNSE1	{#}S	Jump to S** if SE1 event flag is clear.
.	Events - Branch	JNSE2	{#}S	Jump to S** if SE2 event flag is clear.
.	Events - Branch	JNSE3	{#}S	Jump to S** if SE3 event flag is clear.
.	Events - Branch	JNSE4	{#}S	Jump to S** if SE4 event flag is clear.
.	Events - Branch	JNPAT	{#}S	Jump to S** if PAT event flag is clear.
.	Events - Branch	JNFBW	{#}S	Jump to S** if FBW event flag is clear.
.	Events - Branch	JNXMT	{#}S	Jump to S** if XMT event flag is clear.
.	Events - Branch	JNXFI	{#}S	Jump to S** if XFI event flag is clear.
.	Events - Branch	JNXRO	{#}S	Jump to S** if XRO event flag is clear.
.	Events - Branch	JNXRL	{#}S	Jump to S** if XRL event flag is clear.
.	Events - Branch	JNATN	{#}S	Jump to S** if ATN event flag is clear.
.	Events - Branch	JNQMT	{#}S	Jump to S** if QMT event flag is clear.
	Miscellaneous	<empty>	{#}D,{#}S	<empty>
.	Miscellaneous	<empty>	{#}D,{#}S	<empty>
.	Events - Configuration	SETPAT	{#}D,{#}S	Set pin pattern for PAT event. C selects INA/INB, Z selects =/!=, D provides mask value, S provides match value.
alias	Smart Pins	AKPIN	{#}S	Acknowledge smart pins S[10:6]+S[5:0]..S[5:0]. Wraps within A/B pins. Prior SETQ overrides S[10:6].
.	Smart Pins	WRPIN	{#}D,{#}S	Set mode of smart pins S[10:6]+S[5:0]..S[5:0] to D, acknowledge smart pins. Wraps within A/B pins. Prior SETQ overrides S[10:6].
.	Smart Pins	WXPIN	{#}D,{#}S	Set "X" of smart pins S[10:6]+S[5:0]..S[5:0] to D, acknowledge smart pins. Wraps within A/B pins. Prior SETQ overrides S[10:6].
.	Smart Pins	WYPIN	{#}D,{#}S	Set "Y" of smart pins S[10:6]+S[5:0]..S[5:0] to D, acknowledge smart pins. Wraps within A/B pins. Prior SETQ overrides S[10:6].
.	Lookup Table	WRLUT	{#}D,{#}S/P	Write D to LUT address {#}S/PTRx.
.	Hub RAM - Write	WRBYTE	{#}D,{#}S/P	Write byte in D[7:0] to hub address {#}S/PTRx.
.	Hub RAM - Write	WRWORD	{#}D,{#}S/P	Write word in D[15:0] to hub address {#}S/PTRx.
.	Hub RAM - Write	WRLONG	{#}D,{#}S/P	Write long in D[31:0] to hub address {#}S/PTRx. Prior SETQ/SETQ2 invokes cog/LUT block transfer.
alias	Hub RAM - Write	PUSHA	{#}D	Write long in D[31:0] to hub address PTRA++.
alias	Hub RAM - Write	PUSHB	{#}D	Write long in D[31:0] to hub address PTRB++.
.	Hub FIFO - New Read	RDFAST	{#}D,{#}S	Begin new fast hub read via FIFO. D[31] = no wait, D[13:0] = block size in 64-byte units (0 = max), S[19:0] = block start address.
.	Hub FIFO - New Write	WRFAST	{#}D,{#}S	Begin new fast hub write via FIFO. D[31] = no wait, D[13:0] = block size in 64-byte units (0 = max), S[19:0] = block start address.
.	Hub FIFO - New Block	FBLOCK	{#}D,{#}S	Set next block for when block wraps. D[13:0] = block size in 64-byte units (0 = max), S[19:0] = block start address.
.	Streamer	XINIT	{#}D,{#}S	Issue streamer command immediately, zeroing phase.
alias	Streamer	XSTOP		Stop streamer immediately.
.	Streamer	XZERO	{#}D,{#}S	Buffer new streamer command to be issued on final NCO rollover of current command, zeroing phase.

.	Streamer	XCONT {#}D,{#}S	Buffer new streamer command to be issued on final NCO rollover of current command, continuing phase.
.	Branch Repeat	REP {#}D,{#}S	Execute next D[8:0] instructions S times. If S = 0, repeat instructions infinitely. If D[8:0] = 0, nothing repeats.
.	Hub Control - Cogs	COGINIT {#}D,{#}S {WC}	Start cog selected by D. S[19:0] sets hub startup address and PTRB of cog. Prior SETQ sets PTRA of cog.
.	CORDIC Solver	QMUL {#}D,{#}S	Begin CORDIC unsigned multiplication of D * S. GETQX/GETQY retrieves lower/upper product.
.	CORDIC Solver	QDIV {#}D,{#}S	Begin CORDIC unsigned division of {SETQ value or 32'b0, D} / S. GETQX/GETQY retrieves quotient/remainder.
.	CORDIC Solver	QFRAC {#}D,{#}S	Begin CORDIC unsigned division of {D, SETQ value or 32'b0} / S. GETQX/GETQY retrieves quotient/remainder.
.	CORDIC Solver	QSQRT {#}D,{#}S	Begin CORDIC square root of {S, D}. GETQX retrieves root.
.	CORDIC Solver	QROTATE {#}D,{#}S	Begin CORDIC rotation of point (D, SETQ value or 32'b0) by angle S. GETQX/GETQY retrieves X/Y.
.	CORDIC Solver	QVECTOR {#}D,{#}S	Begin CORDIC vectoring of point (D, S). GETQX/GETQY retrieves length/angle.
.	Hub Control - Multi	HUBSET {#}D	Set hub configuration to D.
.	Hub Control - Cogs	COGID {#}D {WC}	If D is register and no WC, get cog ID (0 to 15) into D. If WC, check status of cog D[3:0], C = 1 if on.
.	Hub Control - Cogs	COGSTOP {#}D	Stop cog D[3:0].
.	Hub Control - Locks	LOCKNEW D {WC}	Request a LOCK. D will be written with the LOCK number (0 to 15). C = 1 if no LOCK available.
.	Hub Control - Locks	LOCKRET {#}D	Return LOCK D[3:0] for reallocation.
.	Hub Control - Locks	LOCKTRY {#}D {WC}	Try to get LOCK D[3:0]. C = 1 if got LOCK. LOCKREL releases LOCK. LOCK is also released if owner cog stops or restarts.
.	Hub Control - Locks	LOCKREL {#}D {WC}	Release LOCK D[3:0]. If D is a register and WC, get current/last cog id of LOCK owner into D and LOCK status into C.
.	CORDIC Solver	QLOG {#}D	Begin CORDIC number-to-logarithm conversion of D. GETQX retrieves log {5'whole_exponent, 27'fractional_exponent}.
.	CORDIC Solver	QEXP {#}D	Begin CORDIC logarithm-to-number conversion of D. GETQX retrieves number.
.	Hub FIFO - Read	RFBYTE D {WC/WZ/WCZ}	Used after RDFAST. Read zero-extended byte from FIFO into D. C = MSB of byte. *
.	Hub FIFO - Read	RFWORD D {WC/WZ/WCZ}	Used after RDFAST. Read zero-extended word from FIFO into D. C = MSB of word. *
.	Hub FIFO - Read	RFLONG D {WC/WZ/WCZ}	Used after RDFAST. Read long from FIFO into D. C = MSB of long. *
.	Hub FIFO - Read	RFVAR D {WC/WZ/WCZ}	Used after RDFAST. Read zero-extended 1..4-byte value from FIFO into D. C = 0. *
.	Hub FIFO - Read	RFVARS D {WC/WZ/WCZ}	Used after RDFAST. Read sign-extended 1..4-byte value from FIFO into D. C = MSB of value. *
.	Hub FIFO - Write	WFBYTE {#}D	Used after WRFAST. Write byte in D[7:0] into FIFO.
.	Hub FIFO - Write	WFWORD {#}D	Used after WRFAST. Write word in D[15:0] into FIFO.
.	Hub FIFO - Write	WFLONG {#}D	Used after WRFAST. Write long in D[31:0] into FIFO.
.	CORDIC Solver	GETQX D {WC/WZ/WCZ}	Retrieve CORDIC result X into D. Waits, in case result not ready. C = X[31]. *
.	CORDIC Solver	GETQY D {WC/WZ/WCZ}	Retrieve CORDIC result Y into D. Waits, in case result not ready. C = Y[31]. *
.	Miscellaneous	GETCT D {WC}	Get CT[31:0] or CT[63:32] if WC into D. GETCT WC + GETCT gets full CT. CT=0 on reset, CT++ on every clock. C = same.
.	Miscellaneous	GETRND D {WC/WZ/WCZ}	Get RND into D/C/Z. RND is the PRNG that updates on every clock. D = RND[31:0], C = RND[31], Z = RND[30], unique per cog.
alias	Miscellaneous	GETRND WC/WZ/WCZ	Get RND into C/Z. C = RND[31], Z = RND[30], unique per cog.
.	Smart Pins	SETDACS {#}D	DAC3 = D[31:24], DAC2 = D[23:16], DAC1 = D[15:8], DAC0 = D[7:0].
.	Streamer	SETXFRQ {#}D	Set streamer NCO frequency to D.
.	Streamer	GETXACC D	Get the streamer's Goertzel X accumulator into D and the Y accumulator into the next instruction's S, clear accumulators.
.	Miscellaneous	WAITX {#}D {WC/WZ/WCZ}	Wait 2 + D clocks if no WC/WZ/WCZ. If WC/WZ/WCZ, wait 2 + (D & RND) clocks. C/Z = 0.
.	Events - Configuration	SETSE1 {#}D	Set SE1 event configuration to D[8:0].
.	Events - Configuration	SETSE2 {#}D	Set SE2 event configuration to D[8:0].
.	Events - Configuration	SETSE3 {#}D	Set SE3 event configuration to D[8:0].
.	Events - Configuration	SETSE4 {#}D	Set SE4 event configuration to D[8:0].
.	Events - Poll	POLLINT {WC/WZ/WCZ}	Get INT event flag into C/Z, then clear it.
.	Events - Poll	POLLCT1 {WC/WZ/WCZ}	Get CT1 event flag into C/Z, then clear it.
.	Events - Poll	POLLCT2 {WC/WZ/WCZ}	Get CT2 event flag into C/Z, then clear it.
.	Events - Poll	POLLCT3 {WC/WZ/WCZ}	Get CT3 event flag into C/Z, then clear it.
.	Events - Poll	POLLSE1 {WC/WZ/WCZ}	Get SE1 event flag into C/Z, then clear it.
.	Events - Poll	POLLSE2 {WC/WZ/WCZ}	Get SE2 event flag into C/Z, then clear it.
.	Events - Poll	POLLSE3 {WC/WZ/WCZ}	Get SE3 event flag into C/Z, then clear it.
.	Events - Poll	POLLSE4 {WC/WZ/WCZ}	Get SE4 event flag into C/Z, then clear it.

.	**Events - Poll**	`POLLPAT`		`{WC/WZ/WCZ}`	Get PAT event flag into C/Z, then clear it.
.	**Events - Poll**	`POLLFBW`		`{WC/WZ/WCZ}`	Get FBW event flag into C/Z, then clear it.
.	**Events - Poll**	`POLLXMT`		`{WC/WZ/WCZ}`	Get XMT event flag into C/Z, then clear it.
.	**Events - Poll**	`POLLXFI`		`{WC/WZ/WCZ}`	Get XFI event flag into C/Z, then clear it.
.	**Events - Poll**	`POLLXRO`		`{WC/WZ/WCZ}`	Get XRO event flag into C/Z, then clear it.
.	**Events - Poll**	`POLLXRL`		`{WC/WZ/WCZ}`	Get XRL event flag into C/Z, then clear it.
.	**Events - Poll**	`POLLATN`		`{WC/WZ/WCZ}`	Get ATN event flag into C/Z, then clear it.
.	**Events - Poll**	`POLLQMT`		`{WC/WZ/WCZ}`	Get QMT event flag into C/Z, then clear it.
.	**Events - Wait**	`WAITINT`		`{WC/WZ/WCZ}`	Wait for INT event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.
.	**Events - Wait**	`WAITCT1`		`{WC/WZ/WCZ}`	Wait for CT1 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.
.	**Events - Wait**	`WAITCT2`		`{WC/WZ/WCZ}`	Wait for CT2 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.
.	**Events - Wait**	`WAITCT3`		`{WC/WZ/WCZ}`	Wait for CT3 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.
.	**Events - Wait**	`WAITSE1`		`{WC/WZ/WCZ}`	Wait for SE1 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.
.	**Events - Wait**	`WAITSE2`		`{WC/WZ/WCZ}`	Wait for SE2 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.
.	**Events - Wait**	`WAITSE3`		`{WC/WZ/WCZ}`	Wait for SE3 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.
.	**Events - Wait**	`WAITSE4`		`{WC/WZ/WCZ}`	Wait for SE4 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.
.	**Events - Wait**	`WAITPAT`		`{WC/WZ/WCZ}`	Wait for PAT event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.
.	**Events - Wait**	`WAITFBW`		`{WC/WZ/WCZ}`	Wait for FBW event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.
.	**Events - Wait**	`WAITXMT`		`{WC/WZ/WCZ}`	Wait for XMT event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.
.	**Events - Wait**	`WAITXFI`		`{WC/WZ/WCZ}`	Wait for XFI event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.
.	**Events - Wait**	`WAITXRO`		`{WC/WZ/WCZ}`	Wait for XRO event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.
.	**Events - Wait**	`WAITXRL`		`{WC/WZ/WCZ}`	Wait for XRL event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.
.	**Events - Wait**	`WAITATN`		`{WC/WZ/WCZ}`	Wait for ATN event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.
.	**Interrupts**	`ALLOWI`			Allow interrupts (default).
.	**Interrupts**	`STALLI`			Stall Interrupts.
.	**Interrupts**	`TRGINT1`			Trigger INT1, regardless of STALLI mode.
.	**Interrupts**	`TRGINT2`			Trigger INT2, regardless of STALLI mode.
.	**Interrupts**	`TRGINT3`			Trigger INT3, regardless of STALLI mode.
.	**Interrupts**	`NIXINT1`			Cancel INT1.
.	**Interrupts**	`NIXINT2`			Cancel INT2.
.	**Interrupts**	`NIXINT3`			Cancel INT3.
.	**Interrupts**	`SETINT1`	`{#}D`		Set INT1 source to D[3:0].
.	**Interrupts**	`SETINT2`	`{#}D`		Set INT2 source to D[3:0].
.	**Interrupts**	`SETINT3`	`{#}D`		Set INT3 source to D[3:0].
.	**Miscellaneous**	`SETQ`	`{#}D`		Set Q to D. Use before RDLONG/WRLONG/WMLONG to set block transfer. Also used before MUXQ/COGINIT/QDIV/QFRAC/QROTATE/WAITxxx.
.	**Miscellaneous**	`SETQ2`	`{#}D`		Set Q to D. Use before RDLONG/WRLONG/WMLONG to set LUT block transfer.
.	**Miscellaneous**	`PUSH`	`{#}D`		Push D onto stack.
.	**Miscellaneous**	`POP`	`D`	`{WC/WZ/WCZ}`	Pop stack (K). D = K. C = K[31]. *
.	**Branch D - Jump**	`JMP`	`D`	`{WC/WZ/WCZ}`	Jump to D. C = D[31], Z = D[30], PC = D[19:0].
.	**Branch D - Call**	`CALL`	`D`	`{WC/WZ/WCZ}`	Call to D by pushing {C, Z, 10'b0, PC[19:0]} onto stack. C = D[31], Z = D[30], PC = D[19:0].
.	**Branch Return**	`RET`		`{WC/WZ/WCZ}`	Return by popping stack (K). C = K[31], Z = K[30], PC = K[19:0].
.	**Branch D - Call**	`CALLA`	`D`	`{WC/WZ/WCZ}`	Call to D by writing {C, Z, 10'b0, PC[19:0]} to hub long at PTRA++. C = D[31], Z = D[30], PC = D[19:0].
.	**Branch Return**	`RETA`		`{WC/WZ/WCZ}`	Return by reading hub long (L) at --PTRA. C = L[31], Z = L[30], PC = L[19:0].
.	**Branch D - Call**	`CALLB`	`D`	`{WC/WZ/WCZ}`	Call to D by writing {C, Z, 10'b0, PC[19:0]} to hub long at PTRB++. C = D[31], Z = D[30], PC = D[19:0].
.	**Branch Return**	`RETB`		`{WC/WZ/WCZ}`	Return by reading hub long (L) at --PTRB. C = L[31], Z = L[30], PC = L[19:0].
.	**Branch D - Jump**	`JMPREL`	`{#}D`		Jump ahead/back by D instructions. For cogex, PC += D[19:0]. For hubex, PC += D[17:0] << 2.
.	**Branch D - Skip**	`SKIP`	`{#}D`		Skip instructions per D. Subsequent instructions 0..31 get cancelled for each '1' bit in D[0]..D[31].

.	Branch D - Jump+Skip	SKIPF	{#}D		Skip cog/LUT instructions fast per D. Like SKIP, but instead of cancelling instructions, the PC leaps over them.
.	Branch D - Call+Skip	EXECF	{#}D		Jump to D[9:0] in cog/LUT and set SKIPF pattern to D[31:10]. PC = {10'b0, D[9:0]}.
.	Hub FIFO	GETPTR	D		Get current FIFO hub pointer into D.
.	Interrupts	GETBRK	D	WC/WZ/WCZ	Get breakpoint status into D according to WC/WZ/WCZ. Details not yet documented.
.	Interrupts	COGBRK	{#}D		If in debug ISR, trigger asynchronous breakpoint in cog D[3:0]. Cog D[3:0] must have asynchronous breakpoint enabled.
.	Interrupts	BRK	{#}D		If in debug ISR, set next break condition to D. Else, trigger break if enabled, conditionally write break code to D[7:0].
.	Lookup Table	SETLUTS	{#}D		If D[0] = 1 then enable LUT sharing, where LUT writes within the adjacent odd/even companion cog are copied to this LUT.
.	Color Space Converter	SETCY	{#}D		Set the colorspace converter "CY" parameter to D[31:0].
.	Color Space Converter	SETCI	{#}D		Set the colorspace converter "CI" parameter to D[31:0].
.	Color Space Converter	SETCQ	{#}D		Set the colorspace converter "CQ" parameter to D[31:0].
.	Color Space Converter	SETCFRQ	{#}D		Set the colorspace converter "CFRQ" parameter to D[31:0].
.	Color Space Converter	SETCMOD	{#}D		Set the colorspace converter "CMOD" parameter to D[8:0].
.	Pixel Mixer	SETPIV	{#}D		Set BLNPIX/MIXPIX blend factor to D[7:0].
.	Pixel Mixer	SETPIX	{#}D		Set MIXPIX mode to D[5:0].
.	Events - Attention	COGATN	{#}D		Strobe "attention" of all cogs whose corresponging bits are high in D[15:0].
.	Pins	TESTP	{#}D	WC/WZ	Test IN bit of pin D[5:0], write to C/Z. C/Z = IN[D[5:0]].
.	Pins	TESTPN	{#}D	WC/WZ	Test !IN bit of pin D[5:0], write to C/Z. C/Z = !IN[D[5:0]].
.	Pins	TESTP	{#}D	ANDC/ANDZ	Test IN bit of pin D[5:0], AND into C/Z. C/Z = C/Z AND IN[D[5:0]].
.	Pins	TESTPN	{#}D	ANDC/ANDZ	Test !IN bit of pin D[5:0], AND into C/Z. C/Z = C/Z AND !IN[D[5:0]].
.	Pins	TESTP	{#}D	ORC/ORZ	Test IN bit of pin D[5:0], OR into C/Z. C/Z = C/Z OR IN[D[5:0]].
.	Pins	TESTPN	{#}D	ORC/ORZ	Test !IN bit of pin D[5:0], OR into C/Z. C/Z = C/Z OR !IN[D[5:0]].
.	Pins	TESTP	{#}D	XORC/XORZ	Test IN bit of pin D[5:0], XOR into C/Z. C/Z = C/Z XOR IN[D[5:0]].
.	Pins	TESTPN	{#}D	XORC/XORZ	Test !IN bit of pin D[5:0], XOR into C/Z. C/Z = C/Z XOR !IN[D[5:0]].
.	Pins	DIRL	{#}D	{WCZ}	DIR bits of pins D[10:6]+D[5:0]..D[5:0] = 0. Wraps within DIRA/DIRB. Prior SETQ overrides D[10:6]. C,Z = DIR bit.
.	Pins	DIRH	{#}D	{WCZ}	DIR bits of pins D[10:6]+D[5:0]..D[5:0] = 1. Wraps within DIRA/DIRB. Prior SETQ overrides D[10:6]. C,Z = DIR bit.
.	Pins	DIRC	{#}D	{WCZ}	DIR bits of pins D[10:6]+D[5:0]..D[5:0] = C. Wraps within DIRA/DIRB. Prior SETQ overrides D[10:6]. C,Z = DIR bit.
.	Pins	DIRNC	{#}D	{WCZ}	DIR bits of pins D[10:6]+D[5:0]..D[5:0] = !C. Wraps within DIRA/DIRB. Prior SETQ overrides D[10:6]. C,Z = DIR bit.
.	Pins	DIRZ	{#}D	{WCZ}	DIR bits of pins D[10:6]+D[5:0]..D[5:0] = Z. Wraps within DIRA/DIRB. Prior SETQ overrides D[10:6]. C,Z = DIR bit.
.	Pins	DIRNZ	{#}D	{WCZ}	DIR bits of pins D[10:6]+D[5:0]..D[5:0] = !Z. Wraps within DIRA/DIRB. Prior SETQ overrides D[10:6]. C,Z = DIR bit.
.	Pins	DIRRND	{#}D	{WCZ}	DIR bits of pins D[10:6]+D[5:0]..D[5:0] = RNDs. Wraps within DIRA/DIRB. Prior SETQ overrides D[10:6]. C,Z = DIR bit.
.	Pins	DIRNOT	{#}D	{WCZ}	Toggle DIR bits of pins D[10:6]+D[5:0]..D[5:0]. Wraps within DIRA/DIRB. Prior SETQ overrides D[10:6]. C,Z = DIR bit.
.	Pins	OUTL	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = 0. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	OUTH	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = 1. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	OUTC	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = C. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	OUTNC	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = !C. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	OUTZ	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = Z. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	OUTNZ	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = !Z. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	OUTRND	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = RNDs. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	OUTNOT	{#}D	{WCZ}	Toggle OUT bits of pins D[10:6]+D[5:0]..D[5:0]. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	FLTL	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = 0. DIR bits = 0. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	FLTH	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = 1. DIR bits = 0. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	FLTC	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = C. DIR bits = 0. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	FLTNC	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = !C. DIR bits = 0. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	FLTZ	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = Z. DIR bits = 0. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	FLTNZ	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = !Z. DIR bits = 0. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	FLTRND	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = RNDs. DIR bits = 0. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	FLTNOT	{#}D	{WCZ}	Toggle OUT bits of pins D[10:6]+D[5:0]..D[5:0]. DIR bits = 0. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.

.	Pins	DRVL {#}D {WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = 0. DIR bits = 1. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	DRVH {#}D {WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = 1. DIR bits = 1. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	DRVC {#}D {WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = C. DIR bits = 1. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	DRVNC {#}D {WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = !C. DIR bits = 1. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	DRVZ {#}D {WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = Z. DIR bits = 1. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	DRVNZ {#}D {WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = !Z. DIR bits = 1. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	DRVRND {#}D {WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = RNDs. DIR bits = 1. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Pins	DRVNOT {#}D {WCZ}	Toggle OUT bits of pins D[10:6]+D[5:0]..D[5:0]. DIR bits = 1. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.
.	Math and Logic	SPLITB D	Split every 4th bit of D into bytes. D = {D[31], D[27], D[23], D[19], ...D[12], D[8], D[4], D[0]}.
.	Math and Logic	MERGEB D	Merge bits of bytes in D. D = {D[31], D[23], D[15], D[7], ...D[24], D[16], D[8], D[0]}.
.	Math and Logic	SPLITW D	Split odd/even bits of D into words. D = {D[31], D[29], D[27], D[25], ...D[6], D[4], D[2], D[0]}.
.	Math and Logic	MERGEW D	Merge bits of words in D. D = {D[31], D[15], D[30], D[14], ...D[17], D[1], D[16], D[0]}.
.	Math and Logic	SEUSSF D	Relocate and periodically invert bits within D. Returns to original value on 32nd iteration. Forward pattern.
.	Math and Logic	SEUSSR D	Relocate and periodically invert bits within D. Returns to original value on 32nd iteration. Reverse pattern.
.	Math and Logic	RGBSQZ D	Squeeze 8:8:8 RGB value in D[31:8] into 5:6:5 value in D[15:0]. D = {15'b0, D[31:27], D[23:18], D[15:11]}.
.	Math and Logic	RGBEXP D	Expand 5:6:5 RGB value in D[15:0] into 8:8:8 value in D[31:8]. D = {D[15:11,15:13], D[10:5,10:9], D[4:0,4:2], 8'b0}.
.	Math and Logic	XORO32 D	Iterate D with xoroshiro32+ PRNG algorithm and put PRNG result into next instruction's S.
.	Math and Logic	REV D	Reverse D bits. D = D[0:31].
.	Math and Logic	RCZR D {WC/WZ/WCZ}	Rotate C,Z right through D. D = {C, Z, D[31:2]}. C = D[1], Z = D[0].
.	Math and Logic	RCZL D {WC/WZ/WCZ}	Rotate C,Z left through D. D = {D[29:0], C, Z}. C = D[31], Z = D[30].
.	Math and Logic	WRC D	Write 0 or 1 to D, according to C. D = {31'b0, C).
.	Math and Logic	WRNC D	Write 0 or 1 to D, according to !C. D = {31'b0, !C).
.	Math and Logic	WRZ D	Write 0 or 1 to D, according to Z. D = {31'b0, Z).
.	Math and Logic	WRNZ D	Write 0 or 1 to D, according to !Z. D = {31'b0, !Z).
.	Math and Logic	MODCZ c,z {WC/WZ/WCZ}	Modify C and Z according to cccc and zzzz. C = cccc[{C,Z}], Z = zzzz[{C,Z}].
alias	Math and Logic	MODC c {WC}	Modify C according to cccc. C = cccc[{C,Z}].
alias	Math and Logic	MODZ z {WZ}	Modify Z according to zzzz. Z = zzzz[{C,Z}].
.	Smart Pins	SETSCP {#}D	Set four-channel oscilloscope enable to D[6] and set input pin base to D[5:2].
.	Smart Pins	GETSCP D	Get four-channel oscilloscope samples into D. D = {ch3[7:0],ch2[7:0],ch1[7:0],ch0[7:0]}.
.	Branch A - Jump	JMP #{\}A	Jump to A. If R = 1 then PC += A, else PC = A. "\" forces R = 0.
.	Branch A - Call	CALL #{\}A	Call to A by pushing {C, Z, 10'b0, PC[19:0]} onto stack. If R = 1 then PC += A, else PC = A. "\" forces R = 0.
.	Branch A - Call	CALLA #{\}A	Call to A by writing {C, Z, 10'b0, PC[19:0]} to hub long at PTRA++. If R = 1 then PC += A, else PC = A. "\" forces R = 0.
.	Branch A - Call	CALLB #{\}A	Call to A by writing {C, Z, 10'b0, PC[19:0]} to hub long at PTRB++. If R = 1 then PC += A, else PC = A. "\" forces R = 0.
.	Branch A - Call	CALLD PA/PB/PTRA/PTRB,#{\}A	Call to A by writing {C, Z, 10'b0, PC[19:0]} to PA/PB/PTRA/PTRB (per W). If R = 1 then PC += A, else PC = A. "\" forces R = 0.
.	Math and Logic	LOC PA/PB/PTRA/PTRB,#{\}A	Get {12'b0, address[19:0]} into PA/PB/PTRA/PTRB (per W). If R = 1, address = PC + A, else address = A. "\" forces R = 0.
.	Miscellaneous	AUGS #N	Queue #N[31:9] to be used as upper 23 bits for next #S occurrence, so that the next 9-bit #S will be augmented to 32 bits.
.	Miscellaneous	AUGD #N	Queue #N[31:9] to be used as upper 23 bits for next #D occurrence, so that the next 9-bit #D will be augmented to 32 bits.
.	Instruction Prefix	_RET_ <inst> <ops>	Execute <inst> always and return if no branch. If <inst> is not branching then return by popping stack[19:0] into PC.
.	Instruction Prefix	IF_NC_AND_NZ <inst> <ops>	Execute <inst> if C = 0 and Z = 0.
alias	Instruction Prefix	IF_NZ_AND_NC <inst> <ops>	Execute <inst> if C = 0 and Z = 0.
alias	Instruction Prefix	IF_A <inst> <ops>	Execute <inst> if C = 0 and Z = 0, or if 'above' after a comparison/subtraction.
alias	Instruction Prefix	IF_00 <inst> <ops>	Execute <inst> if C = 0 and Z = 0.
.	Instruction Prefix	IF_NC_AND_Z <inst> <ops>	Execute <inst> if C = 0 and Z = 1.
alias	Instruction Prefix	IF_Z_AND_NC <inst> <ops>	Execute <inst> if C = 0 and Z = 1.
alias	Instruction Prefix	IF_01 <inst> <ops>	Execute <inst> if C = 0 and Z = 1.
.	Instruction Prefix	IF_NC <inst> <ops>	Execute <inst> if C = 0.

alias	**Instruction Prefix**	**IF_AE**	**<inst>**	**<ops>**	Execute <inst> if C = 0, or if 'above or equal' after a comparison/subtraction.
alias	**Instruction Prefix**	**IF_0X**	**<inst>**	**<ops>**	Execute <inst> if C = 0.
.	**Instruction Prefix**	**IF_C_AND_NZ**	**<inst>**	**<ops>**	Execute <inst> if C = 1 and Z = 0.
alias	**Instruction Prefix**	**IF_NZ_AND_C**	**<inst>**	**<ops>**	Execute <inst> if C = 1 and Z = 0.
alias	**Instruction Prefix**	**IF_10**	**<inst>**	**<ops>**	Execute <inst> if C = 1 and Z = 0.
.	**Instruction Prefix**	**IF_NZ**	**<inst>**	**<ops>**	Execute <inst> if Z = 0.
alias	**Instruction Prefix**	**IF_NE**	**<inst>**	**<ops>**	Execute <inst> if Z = 0, or if 'not equal' after a comparison/subtraction.
alias	**Instruction Prefix**	**IF_X0**	**<inst>**	**<ops>**	Execute <inst> if Z = 0.
.	**Instruction Prefix**	**IF_C_NE_Z**	**<inst>**	**<ops>**	Execute <inst> if C != Z.
alias	**Instruction Prefix**	**IF_Z_NE_C**	**<inst>**	**<ops>**	Execute <inst> if C != Z.
alias	**Instruction Prefix**	**IF_DIFF**	**<inst>**	**<ops>**	Execute <inst> if C != Z.
.	**Instruction Prefix**	**IF_NC_OR_NZ**	**<inst>**	**<ops>**	Execute <inst> if C = 0 or Z = 0.
alias	**Instruction Prefix**	**IF_NZ_OR_NC**	**<inst>**	**<ops>**	Execute <inst> if C = 0 or Z = 0.
alias	**Instruction Prefix**	**IF_NOT_11**	**<inst>**	**<ops>**	Execute <inst> if C = 0 or Z = 0.
.	**Instruction Prefix**	**IF_C_AND_Z**	**<inst>**	**<ops>**	Execute <inst> if C = 1 and Z = 1.
alias	**Instruction Prefix**	**IF_Z_AND_C**	**<inst>**	**<ops>**	Execute <inst> if C = 1 and Z = 1.
alias	**Instruction Prefix**	**IF_11**	**<inst>**	**<ops>**	Execute <inst> if C = 1 and Z = 1.
.	**Instruction Prefix**	**IF_C_EQ_Z**	**<inst>**	**<ops>**	Execute <inst> if C = Z.
alias	**Instruction Prefix**	**IF_Z_EQ_C**	**<inst>**	**<ops>**	Execute <inst> if C = Z.
alias	**Instruction Prefix**	**IF_SAME**	**<inst>**	**<ops>**	Execute <inst> if C = Z.
.	**Instruction Prefix**	**IF_Z**	**<inst>**	**<ops>**	Execute <inst> if Z = 1.
alias	**Instruction Prefix**	**IF_E**	**<inst>**	**<ops>**	Execute <inst> if Z = 1, or if 'equal' after a comparison/subtraction.
alias	**Instruction Prefix**	**IF_X1**	**<inst>**	**<ops>**	Execute <inst> if Z = 1.
.	**Instruction Prefix**	**IF_NC_OR_Z**	**<inst>**	**<ops>**	Execute <inst> if C = 0 or Z = 1.
alias	**Instruction Prefix**	**IF_Z_OR_NC**	**<inst>**	**<ops>**	Execute <inst> if C = 0 or Z = 1.
alias	**Instruction Prefix**	**IF_NOT_10**	**<inst>**	**<ops>**	Execute <inst> if C = 0 or Z = 1.
.	**Instruction Prefix**	**IF_C**	**<inst>**	**<ops>**	Execute <inst> if C = 1.
alias	**Instruction Prefix**	**IF_B**	**<inst>**	**<ops>**	Execute <inst> if C = 1, or if 'below' after a comparison/subtraction.
alias	**Instruction Prefix**	**IF_1X**	**<inst>**	**<ops>**	Execute <inst> if C = 1.
.	**Instruction Prefix**	**IF_C_OR_NZ**	**<inst>**	**<ops>**	Execute <inst> if C = 1 or Z = 0.
alias	**Instruction Prefix**	**IF_NZ_OR_C**	**<inst>**	**<ops>**	Execute <inst> if C = 1 or Z = 0.
alias	**Instruction Prefix**	**IF_NOT_01**	**<inst>**	**<ops>**	Execute <inst> if C = 1 or Z = 0.
.	**Instruction Prefix**	**IF_C_OR_Z**	**<inst>**	**<ops>**	Execute <inst> if C = 1 or Z = 1.
alias	**Instruction Prefix**	**IF_Z_OR_C**	**<inst>**	**<ops>**	Execute <inst> if C = 1 or Z = 1.
alias	**Instruction Prefix**	**IF_BE**	**<inst>**	**<ops>**	Execute <inst> if C = 1 or Z = 1, or if 'below or equal' after a comparison/subtraction.
alias	**Instruction Prefix**	**IF_NOT_00**	**<inst>**	**<ops>**	Execute <inst> if C = 1 or Z = 1.
.	**Instruction Prefix**		**<inst>**	**<ops>**	Execute <inst> always. This is the default when no instruction prefix is expressed.