Propeller P8X32A Quick Reference

Returns ROBET (Value) PRITE Symbot ((Coverd)) Prite Symbot ((Coverd))	Spin Language				
### Stocknown (*Parameter 189*) ** Set from PUBPPT method using abort status with optional return value of Cocard (*Parameter 1804*) ** Declare byte sized symbol in VPR block. (*Declare byte siz	-				
### Settler (Locard)		Value	Description		
Country Country Country Country	, ,	✓	ů i		
### Seathwine byte of main memory. ### Seathwine byte sized component of wordforms sized variable. ### Seathwine byte sized component of wordforms sized variable. ### Dytes of main memory with a value. ### Organized on on more region to another in main memory. ### Compare expression against in a single expression of multiple comma-definited depth in the fath of Applications can be a single value (ec. 10) or a range of values (ex. 10.15). ### Dytes of main memory with a value. ### Organized on on more region to another in main memory. ### Compare expression against in a single expression or multiple comma-definited depth in the fath of Applications can be a single value (ec. 10) or a range of values (ex. 10.15). ### Organized on one region to another in main memory. ### Organized on one region to another in main memory. ### Organized on one region to another in main memory. ### Organized on one region to another in main memory. ### Organized on one region to another in main memory. ### Organized on one region to another in main memory. ### Organized on one region to another in main memory. #### Organized on one region to another in main memory. #### Organized on one region to another in main memory. ##### Organized on one region to another in main memory. ##### Organized on one region to another in main memory. ###### Organized on one region to another in main memory. ###################################	y (,				
Sendowthis (Citised)	, , ,		, ,		
### SPERFORM (Devidences, Securities, Count) ### SPERFORM (Devidences, Securities, Count) ### Additict pressors			, ,		
### Statistics (Parameter (Se) *** Additify spression** *** Match Expression** *** Match Expression** *** Statistics (*** Match Expression**) *** Current System Clock frequency in Hz. ** Current System Clock frequency in Hz. ** Current System Clock frequency at run lime. ** Sist or r		-			
CRISE Case-Expression *** Match Expression: *** As Externant(s) *** Current System Clock Requency, in itz. *** Current System Clock Requency, at run time. *** Current System Clock Requency at run time. *** Current Cosy Sin Drawber Volume Frage Content Cosy Sin Drawber Volume Frage Content Cosy Sin Drawber Volume Frage Cost Sin or resident cog by 10 to run Spin code. *** Content Cosy Sin Drawber Volume Volume Frage Cost Sin Or resident cog by 10 to run Propeller Assembly code. *** Content Cosy Sin Drawber Volume Volume Frage Cost Sin Or resident cog by 10 to run Propeller Assembly code. *** Content Cosy Sin Drawber Volume Volume Frage Cost Sin Or resident cog by 10 to run Propeller Assembly code. *** Content Cost Sin Or Repeter Assembly code and get cog 10: 0-7 = succeeded. 1= failed. *** Content Cost Sin Or Repeter Assembly code and get cog 10: 0-7 = succeeded. 1= failed. *** Content Cost Sin Or Repeter Assembly code and get cog 10: 0-7 = succeeded. 1= failed. *** Content Cost Sin Or Repeter Assembly code and get cog 10: 0-7 = succeeded. 1= failed. *** Content Cost Sin Or Repeter Assembly code and get cog 10: 0-7 = succeeded. 1= failed. *** Content Cost Sin Or Repeter Assembly code and get cog 10: 0-7 = succeeded. 1= failed. *** Content Cost Sin Or Repeter Assembly instruction. *** Cost Sin Or Cost Sin			· ·		
** MatchExpression : ** State Internal (S) *			Copy bytes from one region to another in main memory.		
(**I MatchExpression can contain a single expression or multiple comma delimited expressions can be a single expression or multiple comma delimited expressions can be a single expression to be completed where a single expression can be a single expression to be considered. Current Cox for the Propeller Assembly code and expression to condition. CORMENT (Cox find of (Cox find of Expression System Clock frequency in it.) CORNET (Assemble of Cox find of Expression C	→ MatchExpression :				
Expressions can be a single value (ex: 10) or a range of values (ex: 10.15).			Compare expression against matching expression(s), execute code block if match found.		
(-) DTRR:					
Silatement(s) Version number of the Propeller chip			Expressions can be a single value (ex. 10) of a range of values (ex. 1015).		
CLKPREC					
CLINTOIC CLINT (Mode, Frequency) Set both dock mode setting. Set both dock mode setting. Set both dock mode setting. Current 2-bit System Counter value. CURRENT (Congli), SpinMethad ((ParameterLsd)), StackPointer) COGNIT (Cogli), SpinMethad ((ParameterLsd)), StackPointer) COGNIT (Cogli), AsmAddress, Parameter) Start or restart cog by ID to run Propeller Assembly code. COGNIT (Cogli), AsmAddress, Parameter) Start or restart cog by ID to run Propeller Assembly code. CORNEN (SymMethad ((FarameterLsd)), StackPointer) CONNEN (AsmAddress, Parameter) V Start new cog for Spin code and get cog ID-0.7 = succeeded1 = failed. CONNEN (AsmAddress, Parameter) V Start new cog for Propeller Assembly code and get cog ID-0.7 = succeeded1 = failed. CONNEY ((, -1)) Sym = Exp) CON Sym = Exp (((, -1)) Sym = Exp) CON V=Exp ((, -1)) Sym (LOS3) ((((, -1)) *Exp) ((, -1)) Sym (LOS3)) CONSTINT (ConstantExpression) V Declare in-line constant expression to be completely resolved at compile time. CTRR V Counter B Control register. Declare able of data, aligned and sized as specified. Dente (Symbol) (Condition) Instruction Operands (Effect(s)) DET (Symbol) (Condition) Instruction Operands (Effect(s)) DET (Symbol) (Condition) Instruction Operands (Effect(s)) DET (Symbol) (Condition) V Counter B Frequency register. PERCENTIANCE (LESET FOR Contilion(s)) ** Input register for 32-bit ports A. Counter B Frequency register. If and ELSETF each test for TRUE. IFNOT and ELSETFNOT each test for FALSE. ** Escalarment(s) V Counter B Frequency register. If and ELSETF each test for TRUE. IFNOT and ELSETFNOT each test for FALSE. ** Counter B Frequency register. If and ELSETF each test for TRUE. IFNOT and ELSETFNOT each test for FALSE. ** Escalarment(s) ** Counter B Frequency register. ** Fall (Countilion(s) ** Escalarment(s) ** Counter B Frequency register. ** Counter B Frequency register.	CHIPVER	1	Version number of the Propeller chip.		
Set both clock mode and System Clock frequency at run time. ORT	CLKFREQ	✓	Current System Clock frequency, in Hz.		
COSTINT (CogID, SpinMethod ⟨(Parameter Lish), StackPointer) COGINTT (CogID, SpinMethod ⟨(Parameter Lish), StackPointer) COGINTT (CogID, AsmAddress, Parameter) COGNEN (SpinMethod ⟨(Parameter Lish), StackPointer) Start or restart cog by ID for un Propeller assembly code. Start or restart cog by ID for un Propeller (and Interest or CogID) Start now cog for Spin code and get cog ID: 0.7 = succeeded. 1 = failed. COGNEN (SpinMethod ⟨(Parameter Lish), StackPointer) COGNEN (SpinMethod ⟨(Parameter Lish), StackPointer) Start new cog for Propeller Assembly code and get cog ID: 0.7 = succeeded. 1 = failed. COGNEN ((CogID) CON (((□, □)) Sym = Exp) Declare symbolic, global constants. Declare global enumerations (incrementing symbolic constants). Declare global enumerations (incrementing symbolic constants. Declare foliations (incrementing symbolic constants. Declare foliations (incrementing symbolic constants. Declare foliations (incrementing	CLKMODE	✓	Current clock mode setting.		
Content of Cognition Cogn	CLKSET (Mode, Frequency)		Set both clock mode and System Clock frequency at run time.		
Start or restart cog by ID to run Spin code.		✓			
Start or restart cog by ID to run Propeller Assembly code.	COGID	√	Current cog's ID number; 0-7.		
COONEH (SpinMethod ((ParameterList)), StackPointer) COONEH (AsnAddress, Parameter) COONEH (AsnAddress, Parameter) COONEH (AsnAddress, Parameter) COONEH (AsnAddress, Parameter) Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get cog ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get is ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get is ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get is ID: 0-7 = succeeded1 = falled. Start new cog for Spin code and get is ID: 0-7 = succeeded1 = f	COGINIT (CogID, SpinMethod ((ParameterList)), StackPointer)				
COREN (AsmAnddress, Parameter) COR CORD Sign = Exp (((, □)) Sym = Exp) CON Sym = Exp (((, □)) Sym = Exp) ((, □)) **Exp} ((, □)) **Sym([Ols]) CON (*Exp((, □)) Sym (Eols]) ((((, □)) **Exp) ((, □)) Sym([Ols]) CONSTRINT (ConstantExpression) CTRB CTRB CTRB CTRB CTRB COUNTER ConstantExpression CSymbol Alignment (Size) (Data) ([Count]) (, (Size) Data ([Count])) DBT (Symbol Alignment (Size) (Data) ([Count]) (, (Size) Data ([Count])) DBT (Symbol (Condition) Instruction Operands (Effect(s)) DECIAR ([Pin(s]]) TIRE ([Pin(s]]) COUNTER A Frequency register. Counter B Frequency register. Test Condition(s) Test Condition(s) or Test Test Escite State and get its previous state: TRUE or FRLSE. Clock Counter Semphore to furue and get its previous state: TRUE or FRLSE. LOKE Start Address ([Count]) Test Condition (ong size data in DRT block. Test Condition (ong size data in DRT block. Test Condition (ong size data in DRT block. T	COGINIT (CogID, AsmAddress, Parameter)		Start or restart cog by ID to run Propeller Assembly code.		
COGSTOP (CogID) CON Sym = Exp (((, □)) Sym = Exp) CON (#Exp(((, □))) Sym ([Ois]) \((((, □))) #Exp) ((, □)) \$\text{Sym}([Dis])\((((, □))) \((((, □))) *\text{Sym}([Dis])\(((((, □))) *\text{Sym}([Dis])\(((((, □))) *\text{Sym}([Dis])\(((((, □))) *\text{Sym}([Dis])\((((((, □))) *\text{Sym}([Dis])\((((((, □))) *\text{Sym}([Dis])\(((((((, □))) *\text{Sym}([Dis])\((((((((((((((((((((((((((((((((((((
CON Sym = Exp (((, →)) Sym = Exp) CON Sym = Exp (((, →)) Sym = Exp) CON (*Exp (((, →))) Sym (Eo/s]) ((((, →)) *Exp) ((, →)) *Exp) ((, →)) Sym (Eo/s]) CONSTANT (ConstantExpression) CTRB CTRB CTRB COUNTER A Control register. Counter A Control register. Counter B Control register. Counter B Control register. Declare table of data, aligned and sized as specified. Denote Propeller Assembly instruction. (Symbol) (Condition) Instruction Operands (Effect(s)) DTRA ([Pin(s]]) FILE "FileName" Import external file as data in DRT block. FLORT (IntegerConstant) C(Sunter B Frequency register. Counter A Frequency register. Counter A Frequency register. Counter A Frequency register. Counter A Frequency register. Counter B Frequency regis		✓			
Declare symbolic, global constants.	, and the second		Stop cog by its ID.		
(★ Exp ((, →))) Sym ⟨[O/S]⟩ (((, →)) ▶ Exp) ((, →)) Sym ⟨[O/S]⟩ Declare global enumerations (incrementing symbolic constants). CONSTRINT (ConstantExpression) ✓ Declare in-line constant expression to be completely resolved at compile time. CTRB ✓ Counter A Control register. DRT ✓ Counter B Control register. CSymbol A Alignment ⟨Size⟩ ⟨Data⟩ ⟨[Count]⟩ ⟨., Size⟩ Data ⟨[Count]⟩ Declare table of data, aligned and sized as specified. DRT ⟨Symbol⟩ ⟨Condition⟩ Instruction Operands ⟨Effect(s)⟩ Declare table of data, aligned and sized as specified. DRT ⟨Symbol⟩ ⟨Condition⟩ Instruction Operands ⟨Effect(s)⟩ Declare table of data, aligned and sized as specified. DRT ⟨Symbol⟩ ⟨Condition⟩ Instruction Operands ⟨Effect(s)⟩ Declare table of data, aligned and sized as specified. DRT ⟨Symbol⟩ ⟨Condition⟩ Instruction Operands ⟨Effect(s)⟩ Declare table of data, aligned and sized as specified. PERCENTATION Operands (Effect(s)) ✓ Direction register for 32-bit port A. Import external file as data in DRT block. (Convert integer constant expression to compile-time floating-point value in any block (ELSETFORT) (FILE "FNOT) Convert integer constant expression to compile-time floating-point value in any block (ELSETFORT) (FILE "FNOT) Condition(s) ** Estell/Statem	$Sym = Exp \langle ((, \mid \hookrightarrow)) \mid Sym = Exp \rangle \dots$		Declare symbolic, global constants.		
CTRB CTRB COunter A Control register. Counter B Control register. Counter B Control register. Counter B Control register. Declare table of data, aligned and sized as specified. Declare table of data, aligned and sized as specified. Denote Propeller Assembly instruction. DIRR ([Pin(s]]) Counter A Frequency register for 32-bit port A. Import external file as data in DRT block. FLORT (Integer Constant) Counter A Frequency register. Counter A Frequency register. Counter B Frequency register. Counter B Frequency register. Test condition(s) IF and ELSEIF condition(s) IF and ELSEIF each test for TRUE. IFNOT and ELSEIFNOT each test for FALSE. Input register for 32-bit ports A. If and ELSEIF each test for TRUE. IFNOT and ELSEIFNOT each test for FALSE. Input register for 32-bit ports A. Clear semaphore to false and get its previous state: TRUE or FALSE. Check out new semaphore and get its ID: 0-7, or -1 if none were available. LOCKRET (ID) Check out new semaphore and get its previous state: TRUE or FALSE. Check out new semaphore and get its previous state: TRUE or FALSE. Check out new semaphore and get its previous state: TRUE or FALSE. Check out new semaphore and get its previous state: TRUE or FALSE. Check out new semaphore and get its previous state: TRUE or FALSE. Check out new semaphore and get its previous state: TRUE or FALSE. Check out new semaphore and get its previous state: TRUE or FALSE. Check out new semaphore and get its previous state: TRUE or FALSE. Check out new semaphore and get its previous state: TRUE or FALSE. Check out new semaphore and get its previous state: TRUE or FALSE. Check out new semaphore pool in VRB block. Cymbob Long Data ([Count]) Choose FalseAddress] ([Ciffset]) Peclare long-signed and/or long-sized data in DRT block. Check of main memory. Choose Frequency register. Counter B Frequency register. Test condition(s) and execute block of code if valid. Import external file as data in DRT block. Counter B Frequency register. Test condition(s) a	$\langle \# Exp((, \vdash \hookrightarrow)) \rangle Sym \langle [Ofs] \rangle \langle \langle ((, \vdash \hookrightarrow)) \# Exp \rangle ((, \vdash \hookrightarrow)) Sym \langle [Ofs] \rangle \rangle$				
TRB Counter B Control register. Det (Symbol) Alignment ⟨Size⟩ ⟨Data⟩ ⟨[Counf]⟩ ⟨.⟨Size⟩ Data⟨[Counf]⟩⟩ Det (Symbol) ⟨Condition⟩ Instruction Operands ⟨Effect(s)⟩ Denote Propeller Assembly instruction. Denote Propeller Assembly instructi					
DRT \(Symbol) Alignment \(Size\) \(Data\) \([Count]\) \(,\Size\) \(Data\) \([Count]\) \(,\Size\) \(Data\) \([Count]\) \(,\Size\) \(Data\) \([Count]\) \(Declare table of data, aligned and sized as specified.\) Denote Propeller Assembly instruction. Denote Propeller Assemble in DRT block. Denote Tene at its Denote Tene at its Denote Tene at its Denote Tene					
Symboly Alignment Size Data Count] Count] Count] Count] Count] Count] Count] Count Co		*	Counter B Control register.		
Denote Propeller Assembly Instruction.	$\langle Symbol \rangle$ Alignment $\langle Size \rangle$ $\langle Data \rangle$ $\langle [Count] \rangle$ $\langle , \langle Size \rangle$ $Data \langle [Count] \rangle \rangle$		Declare table of data, aligned and sized as specified.		
Import external file as data in DAT block. FLORT (IntegerConstant)	⟨Symbol⟩ ⟨Condition⟩ Instruction Operands ⟨Effect(s)⟩				
FLORT (IntegerConstant) ✓ Convert integer constant expression to compile-time floating-point value in any block FRQR ✓ Counter A Frequency register. FRQB ✓ Counter B Frequency register. Counter B Frequency register. Test condition(s) → I ElselfStatement(s) ⟨ELSEIF Condition(s) → I ElselfStatement(s) ⟨ELSEIF Condition(s) → I ElselfStatement(s) ⟨ELSEIF ach test for TRUE. IFNOT and ELSEIFNOT each test for FRLSE. → I ElseStatement(s) INR ⟨[Pin(s]]⟩ ✓ Input register for 32-bit ports A. LOCKCLR (ID) ✓ Clear semaphore to false and get its previous state: TRUE or FRLSE. LOCKNEN ✓ Check out new semaphore and get its ID: 0-7, or -1 if none were available. Return semaphore back to semaphore pool, releasing it for future LOCKNEN request LOCKSET (ID) ✓ Set semaphore to true and get its previous state; TRUE or FRLSE. LOCKSET (ID) Declare long-sized symbol in YRR block. ⟨Symbol⟩ LONG Data ⟨[Count]⟩ Declare long-sized symbol in YRR block. Cymbol⟩ LONG [BaseAddress] ⟨[Offset]⟩ ✓ Read/write long of main memory. LONGFILL (StartAddress, Value, Count)		· ·	0 1		
FRQB Counter A Frequency register. ((IF IFNOT)) Condition(s) It IStatement(s) (ELSEIF Condition(s) IEstelfStatement(s)) (ELSEIFNOT Condition(s) IEstelfStatement(s)) (ELSE IEstelfStatement(s)) (ELSE IIF and ELSEIF each test for TRUE. IFNOT and ELSEIFNOT each test for FALSE. Input register for 32-bit ports A. Clear semaphore to false and get its previous state; TRUE or FALSE. Check out new semaphore and get its ID; 0-7, or -1 if none were available. Return semaphore back to semaphore pool, releasing it for future LOCKNEM request LOCKSET (ID) LOCKSET (ID) COKSET (ID) Declare long-sized symbol in YRB block. ⟨Symbol⟩ LONG Data ⟨[Count]⟩ Condition(s) and execute block of code if valid. IF and ELSEIFNOT each test for FALSE. IF and ELSEIF each test for TRUE. IFNOT and ELSEIFNOT each test for FALSE. Clear semaphore to false and get its previous state; TRUE or FALSE. Check out new semaphore and get its ID; 0-7, or -1 if none were available. Return semaphore back to semaphore pool, releasing it for future LOCKNEM request to true and get its previous state; TRUE or FALSE. LOCKSET (ID) Declare long-sized symbol in YRB block. Cymbol⟩ LONG Data ⟨[Count]⟩ Declare long-aligned and/or long-sized data in DAT block. LONG [BaseAddress] ⟨[Offset]⟩ Fill longs of main memory. LONGFILL (StartAddress, Value, Count)					
FRQB ((IF IFNOT)) Condition(s) It Is(alterment(s) (ELSEIF Condition(s) I Elsel/IStatement(s) (ELSEIFNOT Condition(s) IF and ELSEIF each test for TRUE. IFNOT and ELSEIFNOT each test for FALSE. IF and ELSEIF each test for TRUE. IFNOT and ELSEIFNOT each test for FALSE. IF and ELSEIF each test for TRUE. IFNOT and ELSEIFNOT each test for FALSE. I Input register for 32-bit ports A. LOCKCER (I/D) ✓ Clear semaphore to false and get its previous state; TRUE or FALSE. LOCKNEH LOCKRET (I/D) Return semaphore back to semaphore pool, releasing it for future LOCKNEW request LOCKSET (I/D) LOCKSET (I/D) ✓ Set semaphore to true and get its previous state; TRUE or FALSE. LOCKSET (I/D) Declare long-sized symbol in VAR block. ⟨Symbol⟩ LONG Data ⟨[Count]⟩ Declare long-sized and/or long-sized data in DAT block. LONG [BaseAddress] ⟨[Offset]⟩ Fill longs of main memory. Fill longs of main memory with a value.					
((IF IFNOT)) Condition(s) →I IfStatement(s) ⟨ELSEIF Condition(s) →I ElselfStatement(s)⟩ ⟨ELSEIFNOT Condition(s) →I ElselfStatement(s)⟩ ⟨ELSEIFNOT Condition(s) →I ElselfStatement(s)⟩ ⟨ELSEIFNOT Condition(s) →I ElselfStatement(s)⟩ ⟨ELSEIFNOT and ELSEIFNOT each test for FRLSE. →I ElseStatement(s)⟩ INR ⟨[Pin(s)]⟩ ✓ Input register for 32-bit ports A. Cockcl (ID) ✓ Clear semaphore to false and get its previous state; TRUE or FRLSE. LOCKNEH LOCKRET (ID) ✓ Check out new semaphore and get its ID; 0-7, or -1 if none were available. Return semaphore back to semaphore pool, releasing it for future LOCKNEH request LOCKSET (ID) ✓ Set semaphore to true and get its previous state; TRUE or FRLSE. LONG Symbol ⟨[Coun(1)]⟩ Declare long-sized symbol in VRR block. ⟨Symbol⟩ LONG Data ⟨[Coun(1)]⟩ Declare long-aligned and/or long-sized data in DAT block. LONG [BaseAddress] ⟨[Offse(1)]⟩ Fill longs of main memory. Fill longs of main memory with a value.			1 3 0		
Test condition(s) ⟨ELSEIF Condition(s) → I ElselfStatement(s)⟩ ⟨ELSEIFNOT Condition(s) → I ElselfStatement(s)⟩ ⟨ELSE → I ElselfStatement(s)⟩ ⟨ELSE → I ElseStatement(s)⟩ ⟨Input register for 32-bit ports A. Clear semaphore to false and get its previous state; TRUE or FALSE. LOCKNEH LOCKNEH ← Check out new semaphore pool, releasing it for future LOCKNEH request lockset (ID) ✓ Set semaphore to true and get its previous state; TRUE or FALSE. LOCKSET (ID) ✓ Set semaphore to true and get its previous state; TRUE or FALSE. LOCKSET (ID) Declare long-sized symbol in VAR block. ⟨Symbol⟩ LONG Data ⟨[Count]⟩ Declare long-aligned and/or long-sized data in DAT block. LONG [BaseAddress] ⟨[Offset]⟩ ✓ Read/write long of main memory. Fill longs of main memory with a value.		-	Counter B Frequency register.		
⟨ELSEIF Condition(s) →1 ElselfStatement(s)⟩ ⟨ELSEIFNOT Condition(s) →1 ElselfStatement(s)⟩ →1 ElselfStatement(s)⟩ ⟨ELSE →1 ElselfStatement(s)⟩ ✓ Input register for 32-bit ports A. LOCKCLR (I/D) ✓ Clear semaphore to false and get its previous state; TRUE or FALSE. LOCKNEW ✓ Check out new semaphore and get its ID; 0-7, or -1 if none were available. LOCKSET (I/D) Return semaphore back to semaphore pool, releasing it for future LOCKNEW request LOCKSET (I/D) ✓ Set semaphore to true and get its previous state; TRUE or FALSE. LONG Symbol ⟨[Count]⟩ ✓ Set semaphore to true and get its previous state; TRUE or FALSE. LONG Symbol ⟨[Count]⟩ ✓ Declare long-sized symbol in VAR block. ⟨Symbol⟩ LONG Data ⟨[Count]⟩ ✓ Read/write long of main memory. LONGFILL (StartAddress, Value, Count) Fill longs of main memory with a value.					
⟨ELSEIFNOT Condition(s) IF and ELSEIF each test for TRUE. IFNOT and ELSEIFNOT each test for FALSE. →	(ELSEIF Condition(s)				
→I ElselfStatement(s)⟩ ⟨ELSE →I ElseStatement(s)⟩ ✓ Input register for 32-bit ports A. LOCKCLR (ID) ✓ Clear semaphore to false and get its previous state; TRUE or FRLSE. LOCKNEH ✓ Check out new semaphore and get its ID; 0-7, or -1 if none were available. LOCKRET (ID) Return semaphore back to semaphore pool, releasing it for future LOCKNEH request LOCKSET (ID) ✓ Set semaphore to true and get its previous state; TRUE or FRLSE. LONG Symbol ⟨[Count]⟩ Declare long-sized symbol in VAR block. ⟨Symbol⟩ LONG Data ⟨[Count]⟩ Declare long-aligned and/or long-sized data in DAT block. LONG [BaseAddress] ⟨[Offset]⟩ ✓ Read/write long of main memory. LONGFILL (StartAddress, Value, Count) Fill longs of main memory with a value.					
⟨ELSE →¹ ElseStatement⟨s⟩⟩ INA ⟨[Pin(s)]⟩ ✓ Input register for 32-bit ports A. LOCKCLR (ID) ✓ Clear semaphore to false and get its previous state; TRUE or FALSE. LOCKNEW ✓ Check out new semaphore and get its ID; 0-7, or -1 if none were available. LOCKRET (ID) Return semaphore back to semaphore pool, releasing it for future LOCKNEW request LOCKSET (ID) ✓ Set semaphore to true and get its previous state; TRUE or FALSE. LONG Symbol ⟨[Count]⟩ Declare long-sized symbol in VAR block. ⟨Symbol⟩ LONG Data ⟨[Count]⟩ Declare long-aligned and/or long-sized data in DAT block. LONG [BaseAddress] ⟨[Offset]⟩ ✓ Read/write long of main memory. LONGFILL (StartAddress, Value, Count) Fill longs of main memory with a value.			IF and Elself each test for Thue. If nut and Elself nut each test for Fhlse.		
TINA ⟨[Pin(s)]⟩ INA ⟨[Pin(s)]⟩ ✓ Input register for 32-bit ports A. Clear semaphore to false and get its previous state; TRUE or FALSE. LOCKNEH LOCKNEH Check out new semaphore and get its ID; 0-7, or -1 if none were available. Return semaphore back to semaphore pool, releasing it for future LOCKNEH request LOCKSET (ID) Set semaphore to true and get its previous state; TRUE or FALSE. LONG Symbol ⟨[Count]]⟩ Declare long-sized symbol in VAR block. ⟨Symbol⟩ LONG Data ⟨[Count]]⟩ Declare long-aligned and/or long-sized data in DAT block. LONG [BaseAddress] ⟨[Offset]]⟩ We Read/write long of main memory. Fill longs of main memory with a value.					
LOCKCLR (ID) ✓ Clear semaphore to false and get its previous state; TRUE or FALSE. LOCKNEW ✓ Check out new semaphore and get its ID; 0-7, or -1 if none were available. Return semaphore back to semaphore pool, releasing it for future LOCKNEW request to semaphore to true and get its previous state; TRUE or FALSE. LOCKSET (ID) ✓ Set semaphore to true and get its previous state; TRUE or FALSE. LONG Symbol ⟨[Count]]⟩ Declare long-sized symbol in VAR block. ⟨Symbol⟩ LONG Data ⟨[Count]]⟩ Declare long-aligned and/or long-sized data in DAT block. LONG [BaseAddress] ⟨[Offset]]⟩ ✓ Read/write long of main memory. Fill longs of main memory with a value.					
LOCKCLR (ID) ✓ Clear semaphore to false and get its previous state; TRUE or FALSE. LOCKNEW ✓ Check out new semaphore and get its ID; 0-7, or -1 if none were available. Return semaphore back to semaphore pool, releasing it for future LOCKNEW request to semaphore to true and get its previous state; TRUE or FALSE. LOCKSET (ID) ✓ Set semaphore to true and get its previous state; TRUE or FALSE. LONG Symbol ⟨[Count]]⟩ Declare long-sized symbol in VAR block. ⟨Symbol⟩ LONG Data ⟨[Count]]⟩ Declare long-aligned and/or long-sized data in DAT block. LONG [BaseAddress] ⟨[Offset]]⟩ ✓ Read/write long of main memory. Fill longs of main memory with a value.	INA ([Pin(s/])	✓	Input register for 32-bit ports A.		
LOCKRET (ID) Return semaphore back to semaphore pool, releasing it for future LOCKNEW request LOCKSET (ID) ✓ Set semaphore to true and get its previous state; TRUE or FALSE. LONG Symbol ⟨[Count]]⟩ Declare long-sized symbol in VAR block. ⟨Symbol⟩ LONG Data ⟨[Count]]⟩ Declare long-aligned and/or long-sized data in DAT block. LONG [BaseAddress] ⟨[Offset]⟩ ✓ Read/write long of main memory. LONGFILL (StartAddress, Value, Count) Fill longs of main memory with a value.	LOCKCLR (ID)	✓			
LOCKSET (ID) ✓ Set semaphore to true and get its previous state; TRUE or FALSE. LONG Symbol ⟨[Count]⟩ Declare long-sized symbol in VAR block. ⟨Symbol⟩ LONG Data ⟨[Count]⟩ Declare long-aligned and/or long-sized data in DAT block. LONG [BaseAddress] ⟨[Offset]⟩ ✓ Read/write long of main memory. LONGFILL (StartAddress, Value, Count) Fill longs of main memory with a value.	LOCKNEW	✓	Check out new semaphore and get its ID; 0-7, or -1 if none were available.		
LONG Symbol ⟨[Count]]⟩ Declare long-sized symbol in VAR block. ⟨Symbol⟩ LONG Data ⟨[Count]]⟩ Declare long-aligned and/or long-sized data in DAT block. LONG [BaseAddress] ⟨[Offset]]⟩ ✓ Read/write long of main memory. LONGFILL (StartAddress, Value, Count) Fill longs of main memory with a value.	LOCKRET (ID)		Return semaphore back to semaphore pool, releasing it for future LOCKNEW requests.		
⟨Symbol⟩ LONG Data ⟨[Count]⟩ Declare long-aligned and/or long-sized data in DAT block. LONG [BaseAddress] ⟨[Offset]⟩ ✓ Read/write long of main memory. LONGFILL (StartAddress, Value, Count) Fill longs of main memory with a value.	LOCKSET (ID)	√			
LONG [BaseAddress] ⟨[Offset]⟩ ✓ Read/write long of main memory. LONGFILL (StartAddress, Value, Count) Fill longs of main memory with a value.	LONG Symbol ([Count])				
LONGFILL (StartAddress, Value, Count) Fill longs of main memory with a value.			Ů Ů		
	LONG [BaseAddress] <[Offset]>	✓	,		
LONCMOVE Doct Addrage Sec Addrage Count	LONGFILL (StartAddress, Value, Count)		,		
	LONGMOVE (DestAddress, SrcAddress, Count)		Copy longs from one region to another in main memory.		
'	LOOKDOWN (Value: ExpressionList)				
	LOOKDOWNZ (Value: ExpressionList)				
	LOOKUP (Index: ExpressionList)	✓	'		
	LOOKUPZ (Index: ExpressionList)	✓			
NEXT Skip remaining statements of REPERT loop and continue with the next loop iteration.	NEXT		Skip remaining statements of REPERT loop and continue with the next loop iteration.		

Spin Language (continued)					
Command	Returns Value	Description			
OBJ $Symbol \ \langle [Count] \rangle : "Object" \ \langle \hookrightarrow Symbol \ \langle [Count] \rangle : "Object" \rangle$		Declare symbol object references.			
OUTA ([Pin(s)])	✓	Output register for 32-bit port A.			
PAR	·	Cog Boot Parameter register.			
PHSA	·	Counter A Phase Lock Loop (PLL) register.			
PHSB	·	Counter B Phase Lock Loop (PLL) register.			
PRI Name ((Par Par\)) \land \: RVal\ \ LVar \([Cnt]\)\\\ LVar \([Cnt]\)\ SourceCodeStatements		Declare private method with optional parameters, return value and local variables.			
PUB Name ((Par Par\)) \land \(\text{RVal} \) \(\text{LVar \[Cnt\]} \rangle LVar \(\text{Cnt\]} \rangle \) SourceCodeStatements		Declare public method with optional parameters, return value and local variables.			
QUIT		Exit from REPERT loop immediately.			
REBOOT		Reset the Propeller chip.			
REPERT ⟨Count⟩ →1 Statement(s)		Execute code block repetitively, either infinitely, or for a finite number of iterations.			
REPERT Variable FROM Start TO Finish ⟨STEP Delta⟩ → Statement(s)		Execute code block repetitively, for finite, counted iterations.			
REPEAT ((UNTIL; WHILE)) Condition(s) →1 Statement(s)		Execute code block repetitively, zero-to-many conditional iterations.			
REPEAT → Statement(s) ((UNTIL: WHILE)) Condition(s)		Execute code block repetitively, one-to-many conditional iterations.			
RESULT	✓	Return value variable for PUB/PRI methods.			
RETURN (Value)	✓	Exit from PUB/PRI method with optional return Value.			
ROUND (FloatConstant)	✓	Round floating-point constant to the nearest integer at compile-time, in any block.			
SPR [Index]	✓	Special Purpose Register array.			
STRCOMP (StringAddress1, StringAddress2)	✓	Compare two strings for equality.			
STRING (StringExpression)	✓	Declare in-line string constant and get its address.			
STRSIZE (StringAddress)	✓	Get size, in bytes, of zero-terminate string.			
TRUNC (FloatConstant)	✓	Remove fractional portion from floating-point constant at compile-time, in any block.			
VAR Size Symbol $\langle [Count] \rangle \langle ((, \vdash \hookrightarrow Size)) \mid Symbol \langle [Count] \rangle \rangle \dots$		Declare symbolic global variables.			
VCFG	✓	Video Configuration register.			
VSCL	✓	Video Scale register.			
WAITCNT (Value)		Pause cog's execution temporarily.			
WAITPEQ (State, Mask, Port)		Pause cog's execution until I/O pin(s) match designated state(s).			
WAITPNE (State, Mask, Port)		Pause cog's execution until I/O pin(s) do not match designated state(s).			
WAITVID (Colors, Pixels)		Pause cog's execution until its Video Generator is available for pixel data.			
WORD Symbol ([Count])		Declare word-sized symbol in VAR block.			
⟨Symbol⟩ WORD Data ⟨[Count]⟩		Declare word-aligned and/or word-sized data in DAT block.			
WORD [BaseAddress] \([Offset] \)	✓	Read/write word of main memory.			
Symbol.WORD <[Offset]>	✓	Read/write word-sized component of long-sized variable.			
WORDFILL (StartAddress, Value, Count)		Fill words of main memory with a value.			
WORDMOVE (DestAddress, SrcAddress, Count)		Copy words from one region to another in main memory.			

	Propeller Assembly Language						
	Instruct	on	Description	Z Result	_ C Result	Result	Clocks
ABS	AValue,	\(*\)SValue	Get absolute value of a number.	Result = 0	S[31]	Written	4
ABSNEG	NValue,	\(* \) SValue	Get the negative of a number's absolute value.	Result = 0	S[31]	Written	4
ADD	Value1,	\(\pm \) Value2	Add unsigned values.	D + S = 0	Unsigned Carry	Written	4
ADDABS	Value,	<#>SValue	Add absolute value to another value.	D + S = 0	Unsigned Carry 3	Written	4
ADDS	SValue1,	<#>SValue2	Add signed values.	D + S = 0	Signed Overflow	Written	4
ADDSX	SValue1,	\(\pm \) SValue2	Add signed values plus C.	Z & (D+S+C = 0)	Signed Overflow	Written	4
ADDX	Value1,	\(\pm \) Value2	Add unsigned values plus C.	Z & (D+S+C = 0)	Unsigned Carry	Written	4
AND	Value1,	\(\pm \) Value2	Bitwise AND values.	Result = 0	Parity of Result	Written	4
ANDN	Value1,	\(\pm \) Value2	Bitwise AND value with NOT of another.	Result = 0	Parity of Result	Written	4
CALL	#Symbol		Jump to address of symbol with intention to return to next instruction.	Result = 0		Written	4
CLKSET	Mode		Set clock mode at run time.			Not Written	823 ¹
CMP	Value1,	\#\>Value2	Compare unsigned values.	D = S	Unsigned (D < S)	Not Written	4
CMPS	SValue1,	\(\pm \) SValue2	Compare signed values.	D = S	Signed (D < S)	Not Written	4
CMPSUB	Value1,	\(\pm \) Value2	Compare unsigned values, subtract second if it is lesser or equal.	D = S	Unsigned (D => S)	Written	4
CMPSX	SValue1,	\$\langle SValue2\$	Compare signed values plus C.	Z & (D = S+C)	Signed (D < S+C)	Not Written	4
CMPX	Value1,	⟨ # ⟩ Value2	Compare unsigned values plus C.	Z & (D = S+C)	Unsigned (D < S+C)	Not Written	4
COGID	Destination		Get current cog's ID.	ID = 0	0	Written	823 ¹
COGINIT	Destination		Re/start cog, ID optional, to run Propeller Assembly or Spin code.	ID = 0	No Cog Free	Not Written	823 1
COGSTOP	CogID		Start a cog by ID.	Stopped ID = 0	No Cog Free	Not Written	823 1
DJNZ	Value,	(#)Address	Decrement value and jump to address if not zero.	Result = 0	Unsigned Borrow	Written	4 or 8 ²

			Propeller Assembly Language	(continued)			
	Instructio		Description	Z Result	C Result	Result	Clocks
HUBOP	Destination,	(#)Operation	Perform a hub operation.	Result = 0		Not Written	823 1
JMP	(#) <i>Address</i>		Jump to address unconditionally.	Result = 0		Not Written	4
JMPRET	RetInstAddr,	(#)DestAddr	Jump to address with intention to "return" to another address.	Result = 0		Written	4
LOCKCLR	ID		Clear semaphore to False and get its previous state.	ID = 0	Prior Lock State	Not Written	823 1
LOCKNEW	NewID		Check out new semaphore and get its ID.	ID = 0	No Lock Free	Written	823 1
LOCKRET	ID		Return semaphore back for future "new semaphore" requests.	ID = 0	No Lock Free	Not Written	823 1
LOCKSET	ID		Set semaphore to true and get its previous state.	ID = 0	Prior Lock State	Not Written	823 1
MAX	Value1,	(#) Value2	Limit maximum of unsigned value to another unsigned value.	S = 0	Unsigned (D < S)	Written	4
MAXS	SValue1,	<#>SValue2	Limit maximum of signed value to another signed value.	S = 0	Signed (D < S)	Written	4
MIN	Value1,	(#) Value2	Limit minimum of unsigned value to another unsigned value.	S = 0	Unsigned (D < S)	Written	4
MINS	SValue1,	(#)SValue2	Limit minimum of signed value to another signed value.	S = 0	Signed (D < S)	Written	4
MOV		⟨ # ⟩ Value	Set register to a value.	Result = 0	S[31]	Written	4
MOVD		\(\psi\) Value	Set register's destination field to a value.	Result = 0		Written	4
MOVI	•	\(\pm \) Value	Set register's instruction field to a value.	Result = 0		Written	4
MOVS	Destination,	\(\pm \) Value	Set register's source field to a value.	Result = 0		Written	4
MUXC	Destination,	⟨ # ⟩ <i>Mask</i>	Set discrete bits of value to state of C.	Result = 0	Parity of Result	Written	4
MUXNC	Destination,	\ /	Set discrete bits of value to state of !C.	Result = 0	Parity of Result	Written	4
MUXNZ		⟨ # ⟩ <i>Mask</i>	Set discrete bits of value to state of !Z.	Result = 0	Parity of Result	Written	4
MUXZ	Destination,	` '	Set discrete bits of value to state of Z.	Result = 0	Parity of Result	Written	4
NEG	NValue,	\$\langle SValue	Get negative of a number.	Result = 0	S[31]	Written	4
NEGC	RValue,	(#) Value	Get value, or its additive inverse, based on C.	Result = 0	S[31]	Written	4
NEGNC	RValue,	\(\psi\) Value	Get value, or its additive inverse, based on !C.	Result = 0	S[31]	Written	4
NEGNZ	RValue,	⟨ # ⟩ <i>Value</i>	Get value, or its additive inverse, based on !Z.	Result = 0	S[31]	Written	4
NEGZ	RValue,	\(\pm \) Value	Get value, or its additive inverse, based on Z.	Result = 0	S[31]	Written	4
NOP			No operation, just elapse four clock cycles.				4
OR	Value1,	(#) <i>Value2</i>	Bitwise OR values.	Result = 0	Parity of Result	Written	4
RCL	Value,	⟨ # ⟩ <i>Bits</i>	Rotate C left into value by specified number of bits.	Result = 0	D[31]	Written	4
RCR	Value,	⟨# ⟩Bits	Rotate C right into value by specified number of bits.	Result = 0	D[0]	Written	4
RDBYTE	Value,	(#) <i>Address</i>	Read byte of main memory.	Result = 0		Written	823 ¹
RDLONG	Value,	(#) <i>Address</i>	Read long of main memory.	Result = 0		Written	823 ¹
RDWORD	Value,	(#) <i>Address</i>	Read word of main memory.	Result = 0		Written	823 1
RET			Return to address.	Result = 0		Not Written	4
REV	Value,	⟨ # ⟩ <i>Bits</i>	Reverse LSBs of value and zero-extend.	Result = 0	D[0]	Written	4
ROL	Value,	⟨ # ⟩Bits	Rotate value left by specified number of bits.	Result = 0	D[31]	Written	4
ROR	Value,	⟨ # ⟩Bits	Rotate value right by specified number of bits.	Result = 0	D[0]	Written	4
SAR	Value,	⟨ # ⟩Bits	Shift value arithmetically right by specified number of bits.	Result = 0	D[0]	Written	4
SHL	Value,	⟨ # ⟩Bits	Shift value left by specified number of bits.	Result = 0	D[31]	Written	4
SHR	Value,	⟨ # ⟩Bits	Shift value right by specified number of bits.	Result = 0	D[0]	Written	4
SUB	Value1,	(#) Value2	Subtract unsigned values.	D - S = 0	Unsigned Borrow	Written	4
SUBABS	Value,	(#)SValue	Subtract absolute value from another value.	D - S = 0	Unsigned Borrow ³	Written	4
SUBS	SValue1,	(#)SValue2	Subtract signed values.	D - S = 0	Signed Overflow	Written	4
SUBSX	SValue1,	(#)SValue2	Subtract signed value plus C from another signed value.	Z & (D-(S+C) = 0)	Signed Overflow	Written	4
SUBX	Value1,	(#) Value2	Subtract unsigned value plus C from another unsigned value.	Z & (D-(S+C) = 0)	Unsigned Borrow	Written	4
SUMC	SValue1,	\$\langle \psi \rangle SValue2\$	Sum signed value with another whose sign is inverted based on C.	D ± S = 0	Signed Overflow	Written	4
SUMNC	SValue1,	(#)SValue2	Sum signed value with another whose sign is inverted based on !C.	D ± S = 0	Signed Overflow	Written	4
SUMNZ	SValue1,	\$\langle \square \square SValue 2	Sum signed value with another whose sign is inverted based on !Z.	D ± S = 0	Signed Overflow	Written	4
SUMZ	SValue1,	(#)SValue2	Sum signed value with another whose sign is inverted based on Z.	D ± S = 0	Signed Overflow	Written	4
TEST	Value1,	(#) Value2	Bitwise AND values to affect flags only.	D = 0	Parity of Result	Not Written	4
TESTN	Value1,	(#) Value2	Bitwise AND value with NOT of another to affect flags only.	D = 0	Parity of Result	Not Written	4
TJNZ	Value,	⟨ # ⟩Address	Test value and jump to address if not zero.	D = 0	0	Not Written	4 or 8 ²
TJZ	Value,	⟨ # ⟩Address	Test value and jump to address if zero.	D = 0	0	Not Written	4 or 8 ²
WAITCHT	Target,	(#)Delta	Pause execution temporarily.	Result = 0	Unsigned Carry	Written	6+
WAITPEQ	State,	(#)Mask	Pause execution until I/O pin(s) match designated state(s).			Not Written	6+
WAITPNE	State,	⟨ # ⟩Mask	Pause execution until I/O pin(s) don't match designated state(s).			Not Written	6+
WAITVID	Colors,	⟨ # ⟩Pixels	Pause execution until Video Generator can take pixel data.	D + S = 0	Unsigned Overflow	Not Written	4+ 4
WRBYTE	Value,	(#)Address	Write byte to main memory.			Not Written	823 1
WRLONG	Value,	(#)Address	Write long to main memory.			Not Written	823 1
WRWORD	Value,	⟨ # ⟩Address	Write word to main memory.			Not Written	823 1
XOR	Value1,	\(\psi\) Value2	Bitwise XOR values.	Result = 0	Parity of Result	Written	4

Hub instructions require 8 to 23 clock cycles to execute depending on the relation between its moment of execution and the cog's hub access window. Cogs receive an "access window" every 16 clocks that hub instructions need to sync to. For a given hub instruction, it will take 0 to 15 clocks to sync plus 8 to execute; 0+8 to 15+8 = 8 to 23 clock cycles.

2 Conditional-Jump instructions take 4 clock cycles if a jump is required and 8 if a jump is not required. They are optimized this way to make loops fast.

3 ADDABS C out: If S is negative, C = the inverse of unsigned borrow (for D-S). SUBABS C out: If S is negative, C = the inverse of unsigned carry (for D+S).

4 WAITVID consumes 4 clocks itself; however, complete data handoff requires 7 clocks (6 at some frequencies) between frames. The combination of CTRA PLL frequency and VSCL

FrameClocks must provide an effective 7 (or 6) system clocks.

Math and					Logi	ic Operators
Operator Constant		Is				
Level ¹	Normal	Assign ²	Expres Integer	sions ³ Float	Unary	Description
		always			✓	Pre-decrement (X) or post-decrement (X).
	++	always			✓	Pre-increment (++X) or post-increment (X++).
Highest	~	always			✓	Sign-extend bit 7 (~X) or post-clear to 0 (X~).
(0)	~~	always			✓	Sign-extend bit 15 (\sim X) or post-set to -1 ($X\sim$).
(0)	?	always			✓	Random number forward (?X) or reverse (X?).
	@	never	✓		✓	Symbol address.
	00	never			✓	Object address plus symbol.
	+	never	✓	✓	✓	Positive (+X); unary form of Add.
	-	if solo	✓	✓	✓	Negate (-X); unary form of Subtract.
	^^	if solo	✓	✓	✓	Square root.
1	- 11	if solo	✓	✓	✓	Absolute value.
	<	if solo	✓		✓	Bitwise: Decode 0 – 31 to long w/single-high-bit.
	>	if solo	✓		✓	Bitwise: Encode long to 0 – 32; high-bit priority.
	!	if solo	✓		✓	Bitwise: NOT.
	<-	<-=	✓			Bitwise: Rotate left.
	->	->=	✓			Bitwise: Rotate right.
2	<<	<<=	✓			Bitwise: Shift left.
_	>>	>>=	✓			Bitwise: Shift right.
	~>	~>=	✓			Shift arithmetic right.
	><	><=	✓			Bitwise: Reverse.
3	&	&=	✓			Bitwise: AND.
4	I	=	✓			Bitwise: OR.
	^	^=	✓			Bitwise: XOR.
	*	*=	✓	✓		Multiply and return lower 32 bits (signed).
5	**	**=	✓	,		Multiply and return upper 32 bits (signed).
	/	/=	✓	✓		Divide (signed).
	//	//=	√			Modulus (signed).
6	+	+=	✓	✓		Add.
	-	-=	√	√		Subtract.
7	#>	#>=	√	√		Limit minimum (signed).
	<#	<#=	√	√		Limit maximum (signed).
	<	<=	√	√		Boolean: Is less than (signed).
	>	>=	✓	√		Boolean: Is greater than (signed).
8	<>	<>=	✓ ✓	✓ ✓		Boolean: Is not equal.
	==	===	✓			Boolean: Is equal.
	=<	=<=		√		Boolean: Is equal or less (signed).
	=>	=>=	✓ ✓	✓ ✓		Boolean: Is equal or greater (signed).
9	NOT	if solo			✓	Boolean: NOT (promotes non-0 to -1).
10	AND	AND=	✓	√		Boolean: AND (promotes non-0 to -1).
. 11	OR	OR=	√	√		Boolean: OR (promotes non-0 to -1).
Lowest	=	always	n/a ³	n/a ³		Constant assignment (CON blocks).
(12)	:=	always	n/a ³	n/a ³	ا علمیان	Variable assignment (PUB/PRI blocks).

¹ Precedence level: higher-level operators evaluate before lower-level operators. Operators in same level are commutable; evaluation order does not matter.

³ Assignment forms of operators are not allowed in constant expressions.

	Assembly	Condition	าร
Condition	Instruction Executes	Condition	Instruction Executes
IF_ALWAYS	always	IF_NC_AND_Z	if C clear and Z set
IF_NEVER	never	IF_NC_AND_NZ	if C clear and Z clear
IF_E	if equal (Z)	IF_C_OR_Z	if C set or Z set
IF_NE	if not equal (!Z)	IF_C_OR_NZ	if C set or Z clear
IF_A	if above (!C & !Z)	IF_NC_OR_Z	if C clear or Z set
IF_B	if below (C)	IF_NC_OR_NZ	if C clear or Z clear
IF_AE	if above/equal (!C)	IF_Z_EQ_C	if Z equal to C
IF_BE	if below/equal (C Z)	IF_Z_NE_C	if Z not equal to C
IF_C	if C set	IF_Z_AND_C	if Z set and C set
IF_NC	if C clear	IF_Z_AND_NC	if Z set and C clear
IF_Z	if Z set	IF_NZ_AND_C	if Z clear and C set
IF_NZ	if Z clear	IF_NZ_AND_NC	if Z clear and C clear
IF_C_EQ_Z	if C equal to Z	IF_Z_OR_C	if Z set or C set
IF_C_NE_Z	if C not equal to Z	IF_Z_OR_NC	if Z set or C clear
IF_C_AND_Z	if C set and Z set	IF_NZ_OR_C	if Z clear or C set
IF_C_AND_NZ	if C set and Z clear	IF_NZ_OR_NC	if Z clear or C clear

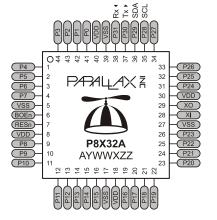
	Constants (pre-defined)						
Constant ¹	Description						
_CLKFREQ	Settable in Top Object File to specify System Clock frequency.						
_CLKMODE	Settable in Top Object File to specify application's clock mode.						
_XINFREQ	Settable in Top Object File to specify external crystal frequency	/ .					
_FREE	Settable in Top Object File to specify application's free space.						
_STACK	Settable in Top Object File to specify application's stack space.						
TRUE	Logical true: -1 (\$FFFFFFF)						
FALSE	Logical false: 0 (\$00000000)						
POSX	Max. positive integer: 2,147,483,647 (\$7FFFFFF)						
NEGX	Max. negative integer: -2,147,483,648 (\$80000000)						
PI	Floating-point PI: ≈ 3.141593 (\$40490FDB)						
RCFAST	Internal fast oscillator: \$00000001 (%0000000000	1)					
RCSLOW	Internal slow oscillator: \$00000002 (%0000000001	.0)					
XINPUT	External clock/oscillator: \$00000004 (%0000000010	0)					
XTAL1	External low-speed crystal: \$00000008 (%0000000100	0)					
XTAL2	External medium-speed crystal: \$00000010 (%0000001000	0)					
XTAL3	External high-speed crystal: \$00000020 (%0000010000	0)					
PLL1X	External frequency times 1: \$00000040 (%0000100000	0)					
PLL2X	External frequency times 2: \$00000080 (%0001000000	0)					
PLL4X	External frequency times 4: \$00000100 (%0010000000	0)					
PLL8X	External frequency times 8: \$00000200 (%0100000000	0)					
PLL16X	External frequency times 16: \$00000400 (%1000000000	0)					

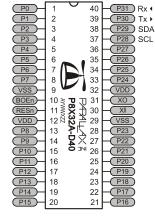
¹ "Settable" constants are defined in Top Object File's CON block. Most expect whole numbers, however _CLKMODE uses Valid Clock Modes, below.

	Valid Clock Modes					
Valid Expression	CLK Reg. Value	Valid Expression	CLK Reg. Value			
RCFAST	0_0_0_00_000	XTAL1 + PLL1X XTAL1 + PLL2X	0_1_1_01_011 0 1 1 01 100			
RCSLOW	0_0_0_00_001	XTAL1 + PLL4X	0_1_1_01_101			
XINPUT	0_0_1_00_010	XTAL1 + PLL8X XTAL1 + PLL16X	0_1_1_01_110 0_1_1_01_111			
XTAL1 XTAL2 XTAL3	0_0_1_01_010 0_0_1_10_010 0_0_1_11_010	XTAL2 + PLL1X XTAL2 + PLL2X XTAL2 + PLL4X XTAL2 + PLL8X XTAL2 + PLL16X	0_1_1_10_011 0_1_1_10_100 0_1_1_10_101 0_1_1_10_110 0_1_1_10_111			
XINPUT + PLL1X XINPUT + PLL2X XINPUT + PLL4X XINPUT + PLL8X XINPUT + PLL16X	0_1_1_00_011 0_1_1_00_100 0_1_1_00_101 0_1_1_00_110 0_1_1_00_111	XTAL3 + PLL1X XTAL3 + PLL2X XTAL3 + PLL4X XTAL3 + PLL8X XTAL3 + PLL16X	0 1 1 11 011 0 1 1 11 100 0 1 1 11 101 0 1 1 11 110 0 1 1 11 111			

Assembly Directives					
Directive	Description				
FIT <i>\langle Address \rangle</i>	Validate previous instr/data fit below an address.				
ORG (Address)	Adjust compile-time cog address pointer.				
⟨Symbol⟩ RES ⟨Count⟩	Reserve next long(s) for symbol.				

Assembly Effects					
Effect	Results In	Effect	Results In		
MC	C Flag modified	WR	Destination Register modified		
WZ	Z Flag modified	NR	Destination Register not modified		





² Assignment forms of binary (non-unary) operators are in the lowest precedence (level 12).